REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION N

11. TITLE (Include Secunty Classification)
A Tool for Automated Validation of Network Protocols

12. PERSONAL AUTHOR(S)
Matthew James Rothlisberger

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 05/89 TO: 08/92	1992, September 24	252

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Communicating Finite State Machines, System of Communicating Machines,Data Link Protocols, ADA, Go_Back_N, Selective Repeat,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis introduces a program that analyzes network protocols using the Communicating Finite State Machines (CFSM) model and the System of Communicating Machines (SCM) model. A simple two machine implementation of CFSM model is initially explored. A number of simple protocols are demonstrated as a means to validate the automated tool (program). The second model implemented is that of SCM model. The SCM tool uses many of the same data structures designed in the CFSM program . The SCM program is validated with an analysis of widely used data link protocols. Both programs were done in the Ada language environment.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. G. M. Lundy	22b. TELEPHONE (Include Area Code): (408) 646-2174	22c. OFFICE SYMBOL: CS/37

Approved for public release; distribution is unlimited

A Tool for Automated Validation of Network Protocols

by

Matthew James Rothlisberger
Captain, United States Army
B. S. Business Administration, Clarion State College , 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

6 August 1992 [i.e. September]

ABSTRACT

This thesis introduces a program that analyzes network protocols using the Communicating Finite State Machines (CFSM) model and the System of Communicating Machines (SCM) model. A simple, two machine implementation of CFSM model is initially explored. A number of simple protocols are demonstrated as a means to validate the automated tool (program).

The second model implemented is that of the SCM model. The SCM tool uses many of the same data structures designed in the CFSM program . The SCM program is validated with an analysis of widely used data link protocols.

Both programs were done in the Ada language environment.

TABLE OF CONTENTS

I. INTRODUCTION 1
II. BACKGROUND OF MODELS 5
A. GENERAL 5
B. COMMUNICATING FINITE STATE MACHINES 7
C. SYSTEM OF COMMUNICATING MACHINES 12
D. LANGUAGE CONSIDERATIONS 18
III. A PROGRAM FOR GENERATING A CFSM REACHABILITY ANALYSIS 21
A. PROGRAM STRUCTURE 21
B. INPUT 24
C. REACHABILITY ANALYSIS 27
D. OUTPUT 31
IV. AN AUTOMATED TOOL FOR SCM REACHABILITY ANALYSIS 34
A. PROGRAM STRUCTURE 35
B. INPUT 39
1. Protocol Variable Definitions 40
2. Predicate-Action Table Representation 42
3. Finite State Machines 45
C. REACHABILITY ANALYSIS 47
1. Global State Analysis 47
2. System State Analysis 52
D. OUTPUT 56
V. AN AUTOMATED ANALYSIS OF SELECTED DATA LINK PROTOCOLS 59
A. CFSM MODEL 59
1. Alternating Bit Protocol 59
2. A More Complex Example: The Sliding Window Protocol. 61
B. SCM MODEL 64
1. Go Back N 64
2. Selective Repeat 74
VI. CONCLUSIONS AND RECOMMENDATIONS 82
APPENDIX A CFSM CODE 84
APPENDIX B SCM CODE 129
APPENDIX C (CFSM) ALTERNATING BIT 172
INPUT (FSM) 172
OUTPUT 173
APPENDIX D (CFSM) SLIDING WINDOW 174
INPUT(FSM) 174
OUTPUT 175
APPENDIX E (SCM) GO_BACK_N, W=1 176
INPUT (FSM) 176

VARIABLE DEFINITIONS 177
PREDICATE-ACTION 178
OUTPUT FORMAT 179
OUTPUT 180
APPENDIX F (SCM) GO_BACK_N, W=2 181
INPUT (FSM) 181
VARIABLE DEFINITIONS 182
PREDICATE-ACTION 183
OUTPUT FORMAT 185
OUTPUT 186
APPENDIX G (SCM) GO_BACK_N, W=3 188
INPUT (FSM) 188
VARIABLE DEFINITIONS 189
PREDICATE-ACTION 190
OUTPUT FORMAT 192
OUTPUT 194
APPENDIX H (SCM) GO_BACK_N, W=4 201
INPUT (FSM) 201
VARIABLE DEFINITIONS 202
PREDICATE-ACTION 203
OUTPUT FORMAT 205
OUTPUT 207
APPENDIX I (SCM) SELECTIVE REPEAT, W=1 228
INPUT 228
VARIABLE DEFINITIONS 229
PREDICATE-ACTION 230
OUTPUT FORMAT 231
OUTPUT 232
APPENDIX J (SCM) SELECTIVE REPEAT, W=2 233
INPUT (FSM) 233
VARIABLE DEFINITIONS 234
PREDICATE-ACTION 235
OUTPUT FORMAT 238
OUTPUT 240
REFERENCES 241
INITIAL DISTRIBUTION LIST 243

LIST OF FIGURES

Figure 1, CFSM, two machine behavior representation 7
Figure 2, CFSM model representation 9
Figure 3, CFSM specification for *stop-and-wait.* 10
Figure 4, CFSM, global reachability analysis, *stop-and-wait.* 11
Figure 5, SCM, two machine behavior representation. 14
Figure 6, SCM, general model representation 15
Figure 7, SCM specification for *stop-and-wait,* finite state machines and variable definitions. 16
Figure 8, SCM, global reachability analysis, *stop-and-wait.* 17
Figure 9, SCM, system reachability analysis, *stop-and-wait.* 17
Figure 10, CFSM run time behavior 22
Figure 11, CFSM compilation units. 23
Figure 12, Finite State Machine representation, *stop_and_wait* 27
Figure 13, CFSM internal reachability graph, *stop_and_wait* 29
Figure 14, CFSM, analysis output, *stop_and_wait* 32
Figure 15, CFSM, analysis output, deadlock/unspecified reception/unexecuted transition example. 33
Figure 16, SCM, specification for *stop_and_wait.* 35
Figure 17, SCM run time behavior. 36
Figure 18, SCM compilation units. 38
Figure 19, SCM, *definitions* package template. 41
Figure 20, SCM, definitions package, *stop_and_wait.* 42
Figure 21, SCM, *Analyze_Predicate* function template. 43
Figure 22, SCM, *Action* procedure template 44
Figure 23, SCM, analyze_predicate.a, *stop_and_wait.* 45
Figure 24, Input File definitions. 46
Figure 25, SCM, Input File, *stop_and_wait.* 46
Figure 26, SCM, global definitions. 49
Figure 27, SCM internal global reachability graph, *stop_and_wait.* 51
Figure 28, SCM, system definitions. 53
Figure 29, SCM internal system reachability graph, *stop_and_wait.* 55
Figure 30, SCM, *output_Gtuple* procedure template 56
Figure 31, SCM, output format, *stop_and_wait.* 57
Figure 32, SCM, analysis output, *stop_and_wait.* 58
Figure 33, CFSM specification, *Alternating Bit.* 60
Figure 34, CFSM specification, *sliding window (w=3)* 62
Figure 35, SCM specification, *Go Back N*, window size of 1..w. 66
Figure 36, SCM, system state analysis, *Go Back N*, w=1 71

Figure 37, SCM, system state analysis, *Go Back N*, w=2..... 72
Figure 38, SCM specification, *Selective Repeat*, window size of 1..w. 75
Figure 39, SCM, system state analysis, *Selective Repeat*, w=1..... 80
Figure 40, SCM, system state analysis, *Selective Repeat*, w=2..... 80

LIST OF TABLES

Table 1, SCM, two machine predicate-action table format. 14
Table 2, SCM specification for *stop_and_wait*, predicate action table. 16
Table 3, CFSM compilation units. 21
Table 4, SCM compilation units. 36
Table 5, SCM, predicate action table, *selective repeat*. 66
Table 6, SCM, predicate action table, *go_back_n*. 75

I. INTRODUCTION

" *simulations are set up to answer the question, what if...*"

-Hamming, R. W., Future Engineering Practice
Course Notes, 11 May 92.

A. BACKGROUND

The past ten years have seen an substantial increase in the need to communicate quickly and reliably over long distances using a wide range of architectures. We, the users, live in a 'come as you are' digital society. Access to some sort of network is needed to move information from user to user. This movement of information takes place on many networks (voice, message, data) and at many levels (physical, data link, network, and higher). The networks in use today tend to be a heterogeneous mix of equipment and protocols. We, the network designers/engineers, must allow the user access to the available resources at the lowest cost. To do this a firm understanding of how machines 'handshake' and talk to one another must be realized. This is accomplish by applying protocol design principles. These principles can be applied to protocols and studied using a wide range of Formal Description Techniques (FDT's). Examples of existing FDT's will be reviewed and the need for an automated set of tools will also be explored.

The need for machines to be able to communicate is inherent in any heterogenous environment. Machines do this through the use of standardized protocols. A protocol is a set of rules that govern the interaction of concurrent processes in distributed systems. Another widely used definition of a protocol is, a set of rules used for communication between two or more processes connected by a communication network. Hand in hand, protocol design and analysis is an important consideration in operating systems, computer networks, and data communications. For a protocol designer/architect to build an appropriate specification, he must use one of a number of modeling techniques.

Models or FDT's of protocols are used for many purposes. They are used to describe the protocol unambiguously so that the exact operation is understood by both the protocol designer, implementor and user. A model is also used to provide a formal framework for a rigorous analysis of the protocol specification. With the advent of internetworking and the birth of Integrated Services Digital Network (ISDN), computer protocols have become more and more complex. The designer must now develop large sets of rules for information exchange that is logically consistent and efficiently implemented. To design a new protocol or to implement an existing one into a computing environment gives rise to a need for such a tool.

There are many formal models available to protocol architects. Some of the more common models include Petri Nets, Communicating Processes, Communicating Finite State Machines (CFSM), System of Communicating Machines (SCM), the Language of Temporal Ordering Specifications (LOTOS), Specification and Description Language (SDL), and Extended State Transition Model Language (ESTELLE). LOTOS and ESTELLE are formal description techniques developed by the International Organization for Standardization (ISO) working laterally with the International Telephone and Telephone Consultative Committee (CCITT).

The ISO is a standards publishing body including the American National Standards Institute (ANSI). In 1980, the ISO saw the advantages of standardizing a hierarchy of protocol services as a reference model for protocol designers. The model includes seven layers: physical, data link, network, transport, session, presentation, and application. The layer/class of protocols that will be analyzed in this thesis are the data link layer protocols.

Each of the models that will be discussed have a means to amplify design principles of communications systems[BART 87]. The first principle is for a model to reflect the behavior of the protocol. Behavior is modeled using conformance models, an example found in [RAND 92]. Secondly, the model must allow refinement by the user. Safety and lively properties should be proven true. And the last principles that must be supported are

those of concurrency and nondeterminism. A good overview is found in King's article[KING 91].

The need for a variety of models is apparent when it is considered that the interaction between machines occurs at different levels in the OSI structure and that behavior can be quite different among levels and machines. The models listed above will be reviewed in the following section. Most of the models have a means to informally follow the design principles. Upon close inspection of each FDT it is apparent that no one is perfect for all applications. As such, an automated tool will be presented that will make the use of two models (CFSM and SCM). The intuitive feel of each protocols coupled with the power of automating such a such models will enable the user to fully enjoy the important design principle of refinement.

B. OBJECTIVES

The objective of this thesis is to present a means of automating two powerful models of protocol validation and analysis. The first that was to be automated is the CFSM model. The data structures and program entities were developed and verified. The second tool automated is the SCM model. Although the SCM used many of the underlying data structures and logic of the CFSM, the SCM model is much more elegant and much more complex. The output information is provided to the user in an intuitive format. Once the two models were fully functional, test cases were input to the models to verify the functionality. Finally, a select number of existing protocols were input and the analysis compared to previous research using a manual method.

C. SCOPE

This thesis presents automated implementation of both the CFSM and SCM models. The implementation of each model was limited to two machine protocols. The specifications covered in this thesis lend themselves very well to the simulation or automated analysis.

The unboundedness properties of CFSM channels are obviously limited to the hardware that the tool is run on. A channel can have a bound the size of the largest machine register, in the case of the test runs, a SUN SPARC station, the upper bound was that of the largest integer(4.294967×10^9 items).

An analysis of select data link protocols are included to illustrate the use of the CFSM and SCM automated models. The specifications will only address procedural rules, not formatting of messages.

D. ORGANIZATION

This thesis is organized into three sections. The first section includes Chapters II and III. Chapters II and III give background information of pertinent models and language considerations. The next section, Chapters IV and V, give a detailed description of how the code was implemented to reflect the behavior of the two models. The final section, Chapter VI, describes the specifications of Alternating Bit, Go_Back_N, and Selective Repeat network protocols. It also describes how the user inputs the information into and receives output from the tool. A means for validating each automated model is discussed in this chapter. Finally, Chapter VII includes conclusions made based on the thesis work and recommendations for future work in the area.

II. BACKGROUND OF MODELS

A. GENERAL

This section contains an overview of some existing FDT's. Each model is a different way to represent a protocol design or reflect network behavior. Each has its own inherent advantages and disadvantages, of which must be considered before application. Two models listed (ESTELLE and LOTOS) are automated.

The first method of description is Petri Nets. Petri Nets are a graphical representation of a systems's states and state changes. The possible states are captured using places which can hold tokens. A particular state is represented by a movement of tokens to states. State changes are described using transitions. This can be visualized as being similar to a directed graph. The input and output arcs associated with each transition determine how token placement changes. The behavior of a system can be determined by examining token movement within the net. Deadlock and freedom of livelock are examined in this model. The complexity of Petri Net representation increases with the size of the protocol being modeled. A major consideration for using this model is the intuitive feel of a protocol is lost on the complex cases.

Another class of FDT are models is called "communicating processes." The following description is more closely examined in Lundy's article[LUND 92b]. The elimination of a set of global states is done through the use of invariants. Rather than generating the set of all possible states, and inspecting them to be sure no undesirable state exists, an assertion is made. The assertion states the desired property. It is proven that the protocol always satisfies the assertion, this must be proven without having to compute all the possible states which might be reached. The communication between processes takes place between unbounded FIFO queues. Processes are emulated by use of variables and statements. The execution of an action is an atomic event and no two actions may occur simultaneously. Since communication between processes can only occur using FIFO queues, actions may

only follow a given sequence. The SCM model demonstrates how this is characteristic is overcome.

Extended State Transition Model Language (ESTELLE) can describe a system in terms of a set of communicating extended finite state machines through use of FIFO channels, similar the to definition of CFSM. This model describes the protocols as a collection of modules, each module is an extended FSM having memory-the difference between an CFSM and Estelle model is that the CFSM model has no memory. Modules of an entity can communicate through FIFO channels[SARI 91]. Messages are exchanged between entities as parameters to the modules. Estelle is based on Pascal and the extension of the language is a feature available to the programmer/user. The models automated in this thesis are similar to Estelle, however, data representation is implemented differently and the power of the language implemented (ADA) is utilized.ESTELLE also allows dynamic module creation/destruction and transition priorities. A model implementation consideration is ESTELLE cannot adequately represent broadcast channels, a shortcoming that the SCM model has shown very well suited for, such as CMSA/CD analysis [LUND 91a].

Specification and Description Language was designed and implemented by groups SGXI and SGX of the CCITT. It was meant as a tool for the design and specification of telephone switches and their underlying protocols. Currently there are two versions of SDL; a graphical tool and a text program tool. Processes are represented by flowcharts, which could be concurrent to other processes. The eight traditional flowchart symbols represent atomic actions such as internal events, input and output, boolean expressions, wait conditions, statements, transitions, and connectors Each flowchart has an associated channel (queue) used to process messages.The Holzman text [HOLZ 91] includes a more specific definition of SDL with some examples. One advantage to this approach is the user gets a feel, graphically, of the behavior of the protocol. The process execution is somewhat restricted to the properties that a FIFO queue has.

The ISO language of Temporal Ordering Specifications (LOTOS) is means of representation using hierarchically structured processes. As with ESTELLE, LOTOS was also developed by the ISO. A hierarchy of processes can correspond to one entity[SIST 91]; a concept that is reflected in object-oriented design environments. Systems represented using this model are organized using a set of interacting processes which exchange information with each other and with the external systems environment through gates. LOTOS is a superset language consisting of an abstract data type language and an algebraic notation language, both of which uphold good design principles covered earlier. Interaction is synchronous through gates that have a one-to-one mapping to interaction points.

The tool implemented in this thesis uses the technique of representing portions of machine behavior as abstract data types, as demonstrated using. A technique of interpreting machine behavior through use of finite state machine representation, as with ESTELLE, will also be integrated into the design. Plans for future upgrade of this tool include a graphical interface similar to that of SDL.

B. COMMUNICATING FINITE STATE MACHINES

One of the first manual tools used for analyzing communication protocol behavior was the communicating finite state machine(CFSM) model. This modeled each machine in the network as a finite automaton, or finite state machine, with communication channels between pairs of machines modeled as one-way, infinite length FIFO queues. There has been a great deal of work in this area, a few include [PENG 91], [VUON 83] and [RUDI 83]. The model is defined for an arbitrary number of machines; however for simplicity sake will be presented as a two machine model as shown in Figure 1.

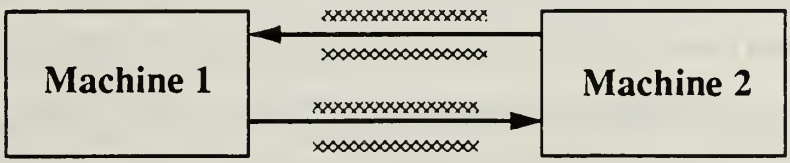


Figure 1: CFSM, two machine behavior representation

In this section the CFSM model will be defined [GOUD 83]followed by a simple protocol analysis to illustrate the model.

A *communicating machine* M is a finite, directed labeled graph with two types of edges, *sending* edges and *receiving* edges. A sending (receiving) edge is labeled '-g' ('+g') for some *message* g, taken from a finite set G of messages. One of the nodes in M is identified as the *initial node*, and each node is reachable from the initial node by some directed path. A node in M whose outgoing edges are all *sending* (*receiving*) edges is a sending (receiving) node; otherwise the node is mixed node. If the outgoing edges of each node in M have distinct labels then M is *deterministic*; otherwise M is *nondeterministic*. The nodes of M are often referred to as *states*; the two terms are used interchangeably.

Let M and N be two communicating machines having the same set G of messages; the pair (M,N) is a network. A *global state* of this network is a four-tuple $[m,c_m,c_n,n]$ where m and n are nodes (states) from M and N, and c_m and c_n are strings from the set G of messages. Intuitively, the global state $[m,c_m,c_n,n]$ means that the machines M and N have reached states m and n, and the communication channels contain the strings c_m and c_n of messages. Channel c_m contains the messages sent from M and N, and channel c_n the messages sent from N to M. The string c_i will be referred to as channel c_i.

The initial global state of (M,N) is $[m_o,E,E,n_o]$, where m_o an n_o are the initial states of M and N, and E is the empty string.

The network progresses as transitions are taken in either M or N. Each transition consists of a state change in one of the machines, and either the addition of a message to the end of one channel (sending transition) or the deletion of a message from the front of one channel (receiving transition).

A sending transition in $M(N)$ adds a message to the end of channel $c_m(c_n)$; a receiving transition in $M(N)$ removes a message from the front of channel $c_n(c_m)$.

If $s_1=[m,c_i,c_j,n]$ is a global state of (M,N), and state s_2 *follows* s_1 if there is a transition (in M or N) which can be executed in s_1, such that the resulting state is s_2. A state s_2 is

reachable from state s_1 if there is a sequence of states $s_i, s_{i+1}, \ldots, s_{i+p}$ such that si follows s_1, s_{i+1} follows s_i, and so on, and s_2 follows s_{i+p}. A state s is reachable if it is reachable from the initial state.

The communication of a network (M,N) is *bounded* if, for every reachable state $[m,c_m,c_n,n]$ there is a nonnegative integer k such that $|c_m| \leq k$ and $|c_n| \leq k$, where $|c|$ denotes the number of messages in channel c.

A reachability graph of a network (M,N) is a directed graph in which the nodes correspond to the reachable global states of (M,N), and the edges represent the *follows* function, such that there is an edge from state s_i to state s_j, if and only if, s_j follows s_i. The edges are labeled with the transition which they represent. The reachability graph can be generated by starting with the initial state, and adding the states which follow it, connecting them to it with edges; and repeating for each new state generated.An overview of the functional units of the CFSM model is shown in Figure 1.

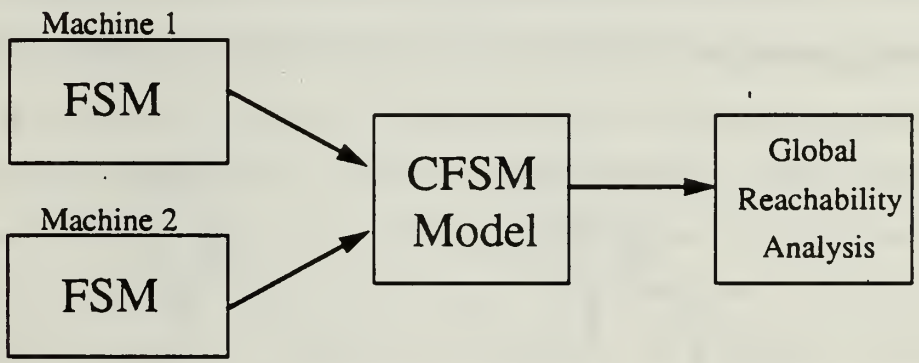


Figure 2: CFSM model representation.

A global state $[m,c_m,c_n,n]$ is a *deadlock state* if both m and n are receiving nodes, and $c_m=c_n=E$, where E denotes the empty string.

A global state $[m,c_m,c_n,n]$ is an *unspecified reception state* if one of the following two conditions are true;

(1) m is a receiving state, the message at the head of channel c_n is g, and none of MS outgoing transitions is labeled '+g.'

(2) n is a receiving state, the message at the head of channel c_m is g, and none of n's outgoing transitions is labeled '+g.'

A simplified version of the *stop-and-wait* data link protocol will be analyzed as an example of analysis with the CFSM model. The interfaces between layer 6 (user)and layer 2 (data link) of the Open Systems Interconnection (OSI) model is transparent in all the examples addressed in this thesis. An assumption is made that the higher layer has passed the information/frames without error. The frames at each layer have accomplished the appropriate concatenation of header and address information. So, at layer 2, this protocol consists of two distinct entities, a sender and a receiver. Machine one serves as the sender and machine 2 serves as the receiver as shown in Figure 3. The sender places a frame on the channel to the receiver. The receiver senses a frame on the incoming channel and accepts the message from the channel, removing the message from the incoming channel. The receiver then sends an acknowledgment packet to the sender. The sender senses the acknowledgment packet and is clear to send another frame of information to the receiver.

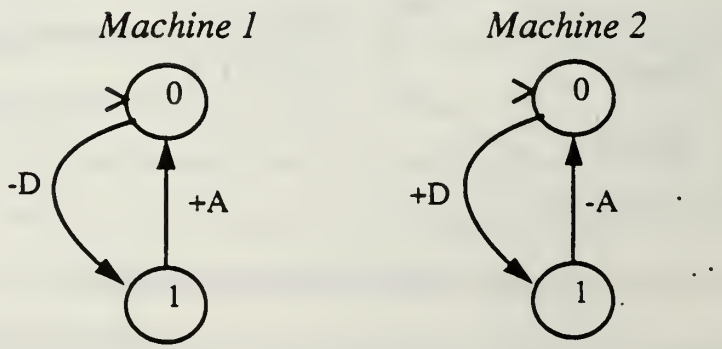


Figure 3: CFSM specification for *stop-and-wait.*

The finite state machines in Figure 3 represent the behavior of the definition of the *stop-and-wait* protocol. The *-D* represents sending data, *+D*, receiving data, *-A*, send

acknowledgment, and $+A$, receive acknowledge. As per the definition of the CFSM model, there is two channels, one from machine 1 to machine 2 and one from machine 2 to machine 1. The notch on state 1 of both machines represents the initial/starting state.

The global reachability analysis graph shown in Figure 4 is free from deadlock, unspecified receptions, and unexecuted transitions.

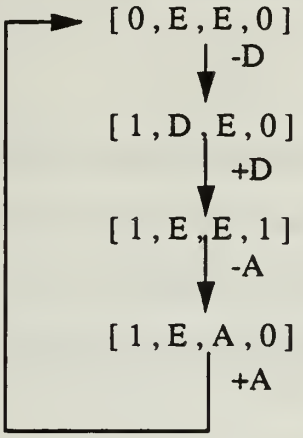


Figure 4: CFSM, global reachability analysis, *stop-and-wait*.

This model has many desirable features as well as some disadvantages that are improved in the SCM model. The one glaring disadvantage is that the analysis might not terminate if the queue length is unbounded. The number of global states in Figure 4 is trivial, but for complex specifications the number of states will lead to a combinatorial state explosion. This is even true when the queue length is very restrictive. As pointed out in [LUND 91b], the specification of a practical protocol can be so complex, containing hundreds of states and transitions, that the user can not be sure of the intended specification or grasp the intuitive feel for what the protocol is intended to do. This model has the advantage of simplicity and a method of analysis that can be easily automated.

C. SYSTEM OF COMMUNICATING MACHINES

In this section the model used to specify and analyze protocols is briefly described. A more detailed description appears in [LUND 91a] . Following the definition of the model will be an analysis of a simple protocol to illustrate the model.

A *system of communicating machines* is an ordered pair $C = (M,V)$, where

$$M=\{m_1,m_2,...,m_n\}$$

is a finite set of *machines*, and

$$V=\{v_1,v_2,...v_n\}$$

is a finite set of *shared variables*, with two designated subsets R_i and W_i specified for each Machine m_i. The subset R_i of V is called the set of *read access variables* for Machine m_i, and the subset W_i the set of *write access variables* for m_i.

Each Machine $m_i \in M$ is defined by a tuple (S_i,s_0,L_i,N_i,τ_i), where

(1) S_i is a finite set of states;

(2) $s_0 \in S_i$ is a designated state called the *initial state* of m_i;

(3) L_i is a finite set of *local variables*;

(4) N_i is a finite set of names, each of which is associated with a unique pair (p,a), where p is a predicate on the variables of $Li \cup Ri$ and a is an *action* on the variables of

$$Li \cup Ri \cup Wi$$

(5) τ_i: $Si \times Ni \rightarrow Si$ is a transition function, which is a partial function from the states and names of m_i to the states of m_i.

Machines model the entities, which in a protocol system are processes and channels. The shared variables are the means of communication between the machines. Intuitively, R_i and W_i are the subsets of V to which m_i has read and write access, respectively. A machine is allowed to make a transition from one state to another when the predicate associated with the name for that transition is true. Upon taking the transition, the action associated with that name is executed.

The set L_i of local variables specifies a name and a range for each. The range must be a finite or countable set of values.

A *system state tuple* is a tuple of all machine states. That is, if (M,V) is a system of n communicating machines, and s_i, for $1 \leq i \leq n$, is the state of Machine m_i, then the n-tuple $(s_1,s_2,...,s_n)$ is the system state tuple of (M,V).

A *system state* is a system state tuple together with its enabled outgoing transitions. Two system states are *equivalent* if every machine is in the same state, *and* the same outgoing transitions are enabled.

The *initial system state* is the system state such that every machine is in its initial state, and the enabled outgoing transitions are the same as in the initial global state.

The *global state* of a system consists of the system state, plus the values of all variables, both local and shared. The *initial global state* is the initial system state, with the additional requirement that all variables have their initial values. A global state *corresponds* to a system state if every machine is in the same state and the same outgoing transitions are enabled.

Let $\tau(s_1,n) = s_2$ be a transition which is defined on Machine m_i. Transition τ is *enabled* if the enabling predicate p, associated with name n, is true. Transition τ may be executed whenever m_i is in state s_1 and the predicate p is true (enabled). The *execution* of τ is an atomic action, in which both the state change and the action a associated with n occur simultaneously. The format for the associated predicate-action table is shown in Table 1.

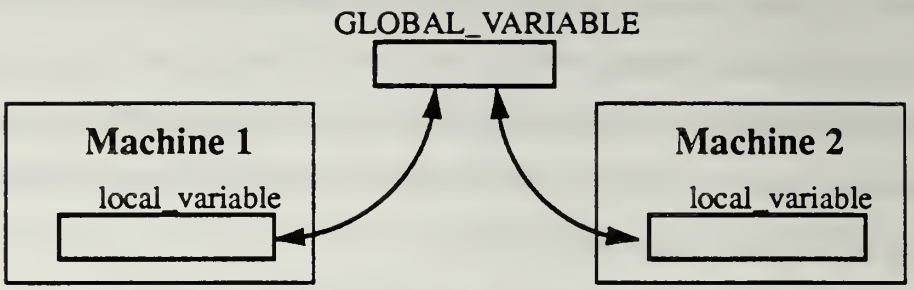


Figure 5: SCM, two machine behavior representation.

TABLE 1: SCM, two machine predicate-action table format.

Transition	Enabling Predicate	Action
Transitions for Machine 1	*Values of variables that must hold true for the transition to be enabled.*	*The local and GLOBAL variable behavior when the transition is taken.*
Transitions for Machine 2	*Same as above.*	*Same as above.*

Note that if the values of all variables are restricted to some finite range, then the model can be reduced to a simple finite state machine. Otherwise an infinite number of global states are possible. However, even if the number of global states is infinite, the number of system states is finite, because of the finiteness of each machine. This may allow a reachability analysis on the system states, when a reachability analysis on the global states

is infinite. General behavior of the SCM model is shown in Figure 1 and the general SCM model representation is found in Figure 1.

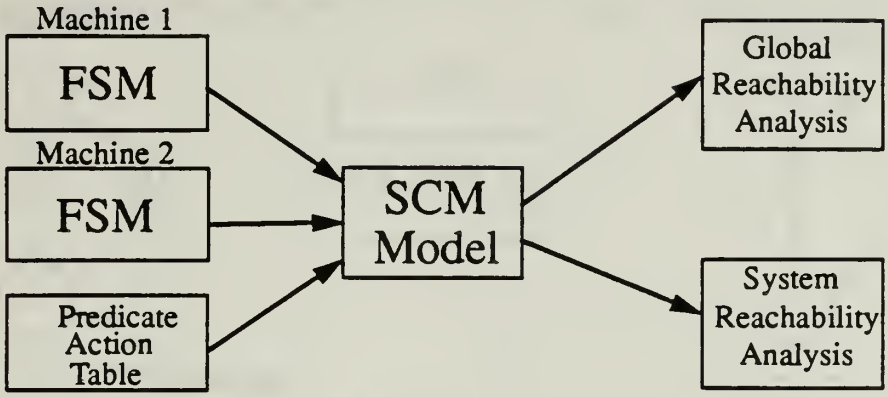


Figure 6: SCM, general model representation

The *stop-and-wait* protocol will also be used to demonstrate the analysis using the SCM model. The stop-and-wait protocol specification is the same as defined in the previous section. The specification as represented by the SCM model is shown as a set of finite state machines and a predicate-action table.

The finite state machine representation for the SCM model is similar to the CFSM example. Again this protocol is only demonstrated with two machines. The FSM's are shown in Figure 3. Also shown are the local and global variables. The local variables in Machine 1 and 2 can have the values of D(data), A (acknowledgment), and E(empty). The

initial value for *out_buff* is D and the initial values for all other variables is E. The system global variable, *CHAN* can have the same values as the local variables.

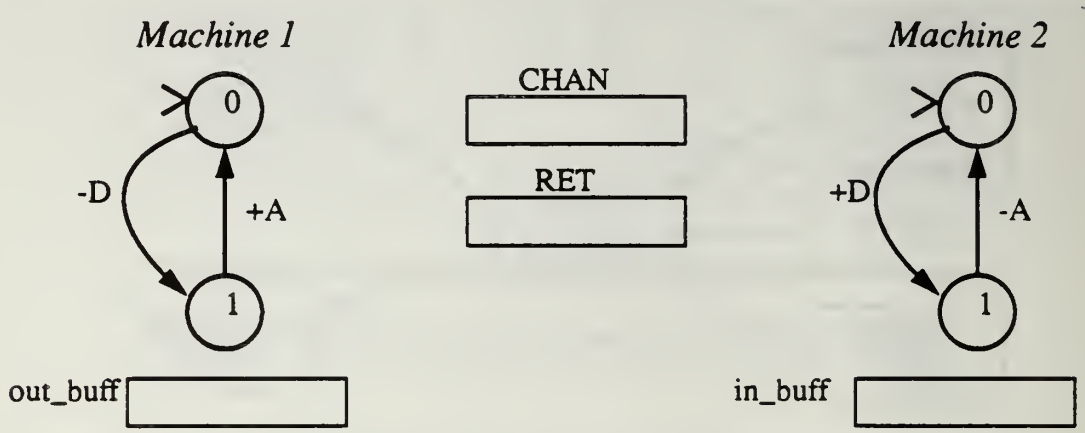


Figure 7: SCM specification for *stop-and-wait*, finite state machines and variable definitions.

The predicate-action table is shown in Table 2. For this example the assumption is made that data is always made available to the CHAN from *out_buff*.

TABLE 2: SCM specification for *stop_and_wait*, predicate action table.

Transition	Enabling Predicate	Action
-D	*CHAN = E* ∧ *out_buff /= E*	*CHAN := out_buff*
+A	*RET = A*	*RET := E* *CHAN := E*
+D	*CHAN /= E*	*in_buff := CHAN*
-A	*true*	*RET := A* *in_buff := e;*

The global state reachability and system state reachability graphs are found in Figure 4 and Figure 4. The format for the global state tuple of the *stop-and-wait* protocol is:

[Machine1_state, out_buff, CHAN, RET, in_buff, Machine2_state]

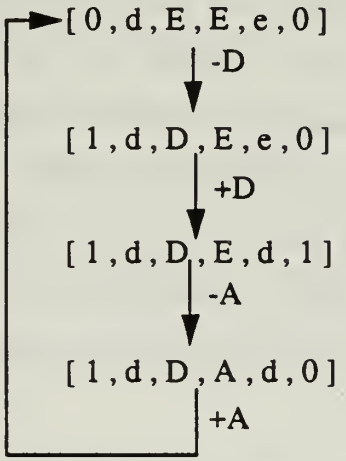


Figure 8: SCM, global reachability analysis, *stop-and-wait*.

The format for a system state tuple for all cases of analysis is:

[Machine1_state, Machine2_state]

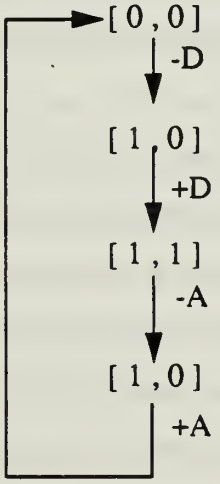


Figure 9: SCM, system reachability analysis, *stop-and-wait*.

The SCM model has desirable properties found in the CFSM model as well as overcoming some of the inherent disadvantages of the CFSM model. In the SCM model the behavior of the protocol can be clearly and quite adequately represented, maintaining an intuitive feel of the specification. The SCM model can ameliorate the combinatorial state explosion through the use of system state analysis, greatly reducing the generated states. Instead of implicit queues, shared variables are used for communications between processes. This allows communications between machines in non sequential manner, unlike a FIFO queue representation in the CFSM model.

The final advantage is the nature of the SCM's representation of a protocol gives it the feel of a programming language. Although more complex to program than a CFSM model, the actions associated with the FSM and the predicate-action table lend themselves to automated implementation.

D. LANGUAGE CONSIDERATIONS

Which language should the CFSM and SCM models would be implemented in? Before all the available languages were researched, a list of desirable properties that the language must have (specific to the models), was developed. After a close inspection of the definition and nuances of the CFSM model, SCM model, and the reachability analysis generated, there were a number of language properties that were found desirable to this project.

The language properties should support hardware and software design issues. The code must be portable from one architecture to another. The language should have a means to create different class instances from a base class. An intuitive means to provide meaningful output of the analysis and programming error messages to the user enhances the program's utility. Since the program must simulate network specifications there is an inherent need to be able to do multiprocessing or multitasking in the programming environment. The language of choice should enforce the rules of strong typing, that is not

allowing mixing of types and subtypes. The final property of the language should be its ease of use and understandibility.

The language of choice should be portable between different machines. It cannot be assumed that the user has access to a mainframe computer or workstation. The language should be compilable on a machine as small as a personal computer.

Implementation of the models should help the user to avoid and detect mistakes. The environment should prompt the user when a syntactical or semantic error is made. The error messages should be meaningful. Inherent to this requirement, the language should enforce strict definitions of atomic structures, such as data structures.

Dynamic list creation /deletion are necessary in reachability graph construction. This allows flexible and ultimately limitless (hardware specific) analyses to be done. Linked list creation and traversal should make use of reusable programming units. The logic for creating new nodes should allow the program to 'remember' where the last node was built. This should be done automatically, without user intervention after compile time.

An important property, although subject to varying opinion, is ease of use. The project is developed in one language, but the human interface to the underlying code must be understandable and intuitive. Hand in hand with ease of use, is ease of maintainability. There should be enough on-line and off-line help to allow the user to navigate the the user interface. An understandable debugger was also a factor in the choice.

Ada was chosen because it supports the above mentioned properties. It is a language that is portable between different architectures. It supports generic class creation and instantiation. Through the use of predefined input and output packages, the user is allowed to build a suitable interface environment. With the use of exception handling meaningful error messages can be created and employed. Ada also has the ability to multitask, simulating parallel processing. Finally, Ada is easy to use. The code can be read by a novice and understand what is meant to happen.

The language of C/C++ was not chosen due to a few limitations. At the time of this writing it could not support multitasking needed to simulate concurrency or broadcast

networks. It was also difficult to do generic-like coding. The object orientation of the design lent itself very nicely to the structures used in the CFSM and SCM model as covered in [RUMB 91]. It became apparent that there was reused code that would have been more efficiently implemented with generic data structures. Although it could have been done with the use of macro-like instructions, generic packages made the project more compact and efficient. Generic package creation and instantiation was not supported by the current version of the C/C++ compiler. The C programming environment does not support exception handling; programming error detection messages were vague and could not be developed by the user. A good means of automatic implementation of error messages in the C++ environment was not available at the time of this publishing. Ada could do this through use of exception handling.

III. A PROGRAM FOR GENERATING A CFSM REACHABILITY ANALYSIS

In this chapter the organization of the CFSM program will be described.The means for input, output, and reachability analysis will be highlighted. Excerpts of the underlying code will be accompanied by a brief explanation. The formal definition of the CFSM model found in Chapter II is the basis for constructing the program.

A. PROGRAM STRUCTURE

The structure of the CFSM program is based on functional units (objects) of the general CFSM model. The data structures of the basic objects must represent communication channels, machine states, transitions, and a means for capturing global tuple (state) values. In addition to constructing the fundamental data structures, there must be an intuitive input mechanism for the FSM's and an understandable display of the analysis.

Implementation details should be hidden from the user. Operations such as loading the CFSM into memory, performing a reachability analysis, constructing the global reachability graph, and traversing the graph during searches/output are independent of the specific protocol to be analyzed.

The program consists of input related procedures, a reachability analysis, and output procedures. To help manage such a complex and large programming project, separate compilation units were used. The compilation units were physically grouped by file according to the function it performed as shown below:

TABLE 3: CFSM compilation units.

Compilation Unit	Description	File
read_in_file	parse text input file	input.a
load_machine_array	builds machine adjacency lists from parsed file	input.a

Compilation Unit	Description	File
build_Gstate_graph	builds global reachability analysis graph	reachability.a
clear_pointers	clears values for another input file	reachability.a
search_for_tuple	performs BFS search of graph	search.a
IsEqual	compares global records for equality (similar to =)	search.a
output_Gstate_node	format for node output	output.a
output_Gstate_transition	format for transition output	output.a
output_Gstates	traverses graph and outputs nodes and transitions	output.a
output_machine_arrays	format output of contents of adjacency lists	output.a
create_output_file	creates file for analysis output.	output.a

This use of separate subprograms(compilation units) facilitated the development of the SCM program from existing CFSM code.

The behavior at run time is shown in Figure 10 and associated files of the CFSM program are shown in Figure 11.

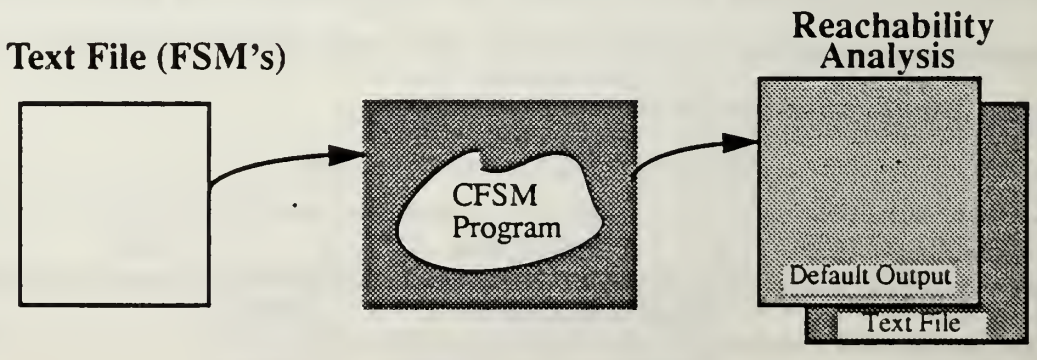


Figure 10: CFSM run time behavior

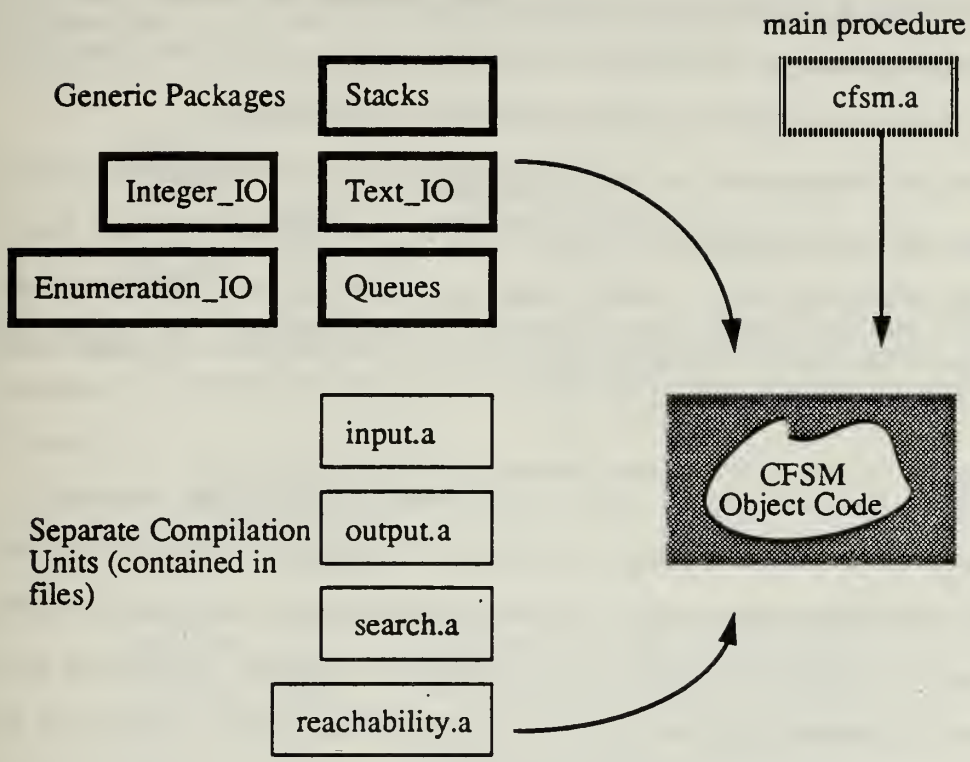


Figure 11: CFSM compilation units.

During the design phase of the program it became apparent that some software components and structures were used more than once. For instance, when doing a reachability analysis many types of stacks and queues were used. Although the underlying data types were different, the algorithm for each structure was exactly the same. To increase efficiency, generic packages were used. Generic units are defined as a reusable software module or a program unit template [GONZ 91].

The implementation of *stacks* and *queues* is accomplished using generics. For instance, within the program there is a need for a queue of characters representing the flow of information on the channels between two machines. To assist in the construction of the reachability graph there needs to be a queue of pointers to graph nodes (see Section C). Each type of queue has some common procedures and functions. Each needs procedures to clear the queue, enqueue, and dequeue. Each must also have functions that return the value

of the first item of the queue, determine if the queue is empty and determine if the contents of two queues are equal. If these common functions and procedures had to be rewritten with a different underlying data type, the number of compilation units as well as the object code would increase; thus the user would be saddled with more 'waiting' time.

The protocol environment can be modified by using generic parameters. The generic package, *queues*, has two parameters to the object- the item type and the maximum size of the queue. This allows the user to define what type of items are contained in the channel(queue) and how big the channel(queue) can be. Two instantiations of *queues* in the program are:

```
package queue_pack is new queues(character, MAX=>3);
package Gpointer_queue_pack is new queues(Glink_type,MAX=>10);
```

The *queue_pack* package defines a queue of characters. A ceiling or bound can be placed on the amount of messages on a channel. If an unbounded channel is to be simulated the maximum allowable integer can be given. The pointer queue *Gpointer_queue_pack* gives the user a means to determine the maximum size of a reachability graph. Although, in the general case, a large number is preferred to allow all tuples (states) to be generated in a protocol reachability graph. The generic package *stacks* was implemented in a similar fashion.

B. INPUT

An important step in designing the CFSM and SCM programs is developing a meaningful method of inputting the finite state machines. The graphical representation of a simple FSM conveys a behavior associated with a protocol specification. A means to transfer this graph into a data structure that can be used in the reachability analysis was developed.

The FSM's were input as a text file. This file is built by the user with a set of language rules similar to Backus-Naur Form (BNF). The input file is parsed one line at a time. Each line is read into a line buffer and tokens formed according to the rules defined below. From

the tokens, an internal data structure is generated to represent the set of finite state machines. The list of valid instructions for finite state machine input is:

```
start
machine          <natural>
state            <natural>
initial_state    <natural> <natural>
trans            <-|+> <a|b|...|z|A|B|...|Z> <natural>
finish
```

The tokens are cast into either enumerated types (instructions) or integers (integer variables). The integer variables have been formally defined within the main procedure in Appendix A.

The meaning of the instructions are:

```
start            Serves as a beginning flag for the file.
machine          Defines the current machine.
state            Defines the current state.
initial_state    The intitial/start state for
                 machines one and two.
trans            Transition type, transition
                 message, and next state.
finish           This token serves as an ending flag for
                 the file.
```

Representation of a finite state machine using the above convention has some inherent constraints. Since an input token, such as the transition *-D*, cannot be directly caste into an enumeration token (no special characters at the beginning of a token), the (-,+) must be converted separately to (snd,rcv) tokens. The use of alphabetic characters to represent messages in a channel, limits messages to 52 distinct types (a..z,A..Z). The input file for *stop_and_wait* is:

```
start
machine 1
state 0
trans -D 1
state 1
trans +A 0
machine 2
state 0
trans +D 1
state 1
trans -A 0
initial_state 0 0
finish
```

The data structure representing the CFSM is then used to construct a reachability analysis graph. The two data structures that support directed graphs (or finite state machines) are adjacency lists and adjacency matrices. Since the use of adjacency matrices to construct directed graphs can lead to wasted hardware memory, adjacency lists (one dimensional array of linked lists) were implemented.

The data structure to build the adjacency list and the constraints are:

```
type machine_array_record_type;
type Mlink_type is access machine_array_record_type;
type cfsm_transition_type is (snd,rcv,unused);
type executed_type is (yes,no);
type machine_array_record_type is
      record
            transition        : cfsm_transition_type;
            message           : character;
            next_state        : natural;
            executed          : executed_type;
            Mlink             : Mlink_type
end record;

type machine_array_type is array(positive range<>)
      of Mlink_type;
type system_array_type is array(1..2)
      of machine_array_type;
```

Some data structures shown above are peculiar to Ada. Access types are data types that provide an access ("pointer") to an object of another type or subtype. It reserves storage locations during the execution of a program dynamically by use of a memory allocator. A record type is simply a collection of elements where each element is referred to by its name.

The array of linked lists is defined as an unconstrained array; whereas, at compile time, the number of machines is set at two. To illustrate the finite state machine data structure the CFSM *stop_and_wait* protocol is shown in Figure 12.

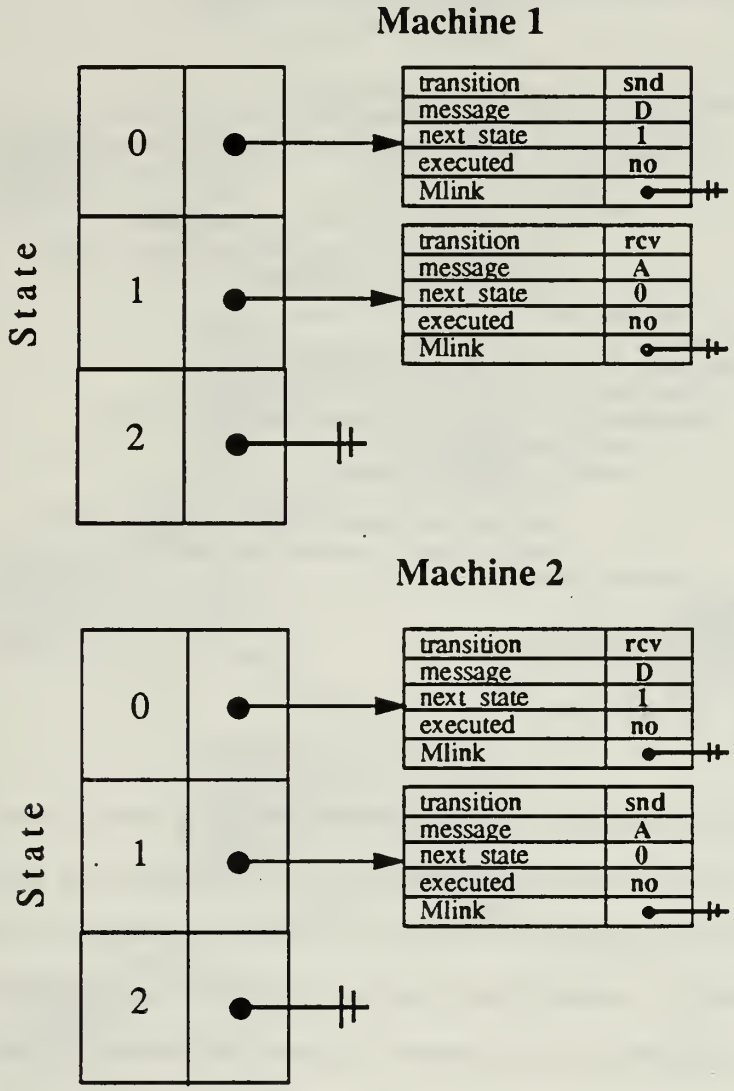


Figure 12: Finite State Machine representation, *stop_and_wait*

C. REACHABILITY ANALYSIS

In order to determine if all states in a network are reachable a graph is constructed. After the textual representation of the CFSM is input, the adjacency lists are constructed as described in the previous section. The initial states for each machine indicate the starting

position of each list. From the lists a directed graph is constructed. It is from this directed graph that deadlocks and unspecified receptions are sensed and the appropriate output message is displayed.

The algorithm to construct the global reachability graph is:

```
loop
    for machine1 array_index in 1..rowsize loop
        if machine1 (array_index).transition=snd or
        machine1(array_index).message=top_of.queue21 then
            make temp Gstate record
            search list for Gstate record
                if found then link current to found state
                else make new node and link into Gstate_graph
                 and push pointer onto pointer stack
        else none_found
    end loop
    for machine2  array_index in 1..rowsize loop
        if machine2 (array_index).transition=snd or
        machine2(array_index).message=top_of.queue12 then
            make temp Gstate record
            search list for Gstate record
                if found then link current to found state
                else make new node and link into Gstate_graph
                 and push pointer onto pointer stack
        else none_found
    end loop
    if stack is empty then
        raise STACK_EMPTY
    else
        pop last Gstate
end loop
```

The initial global state tuple (node) is created from the starting state of each machine's adjacency list. From the top node, tuples (global states) are added to the graph using the reachability algorithm. The algorithm shows the graph being constructed with stack based implementation, allowing a breadth first construct. The option is given to the user to construct the graph depth first. A case statement is used to toggle between stack or queue procedures/functions (this is not shown in the algorithm above.) Figure 13 shows the internal representation of the global reachability graph for the *stop_and_wait* protocol

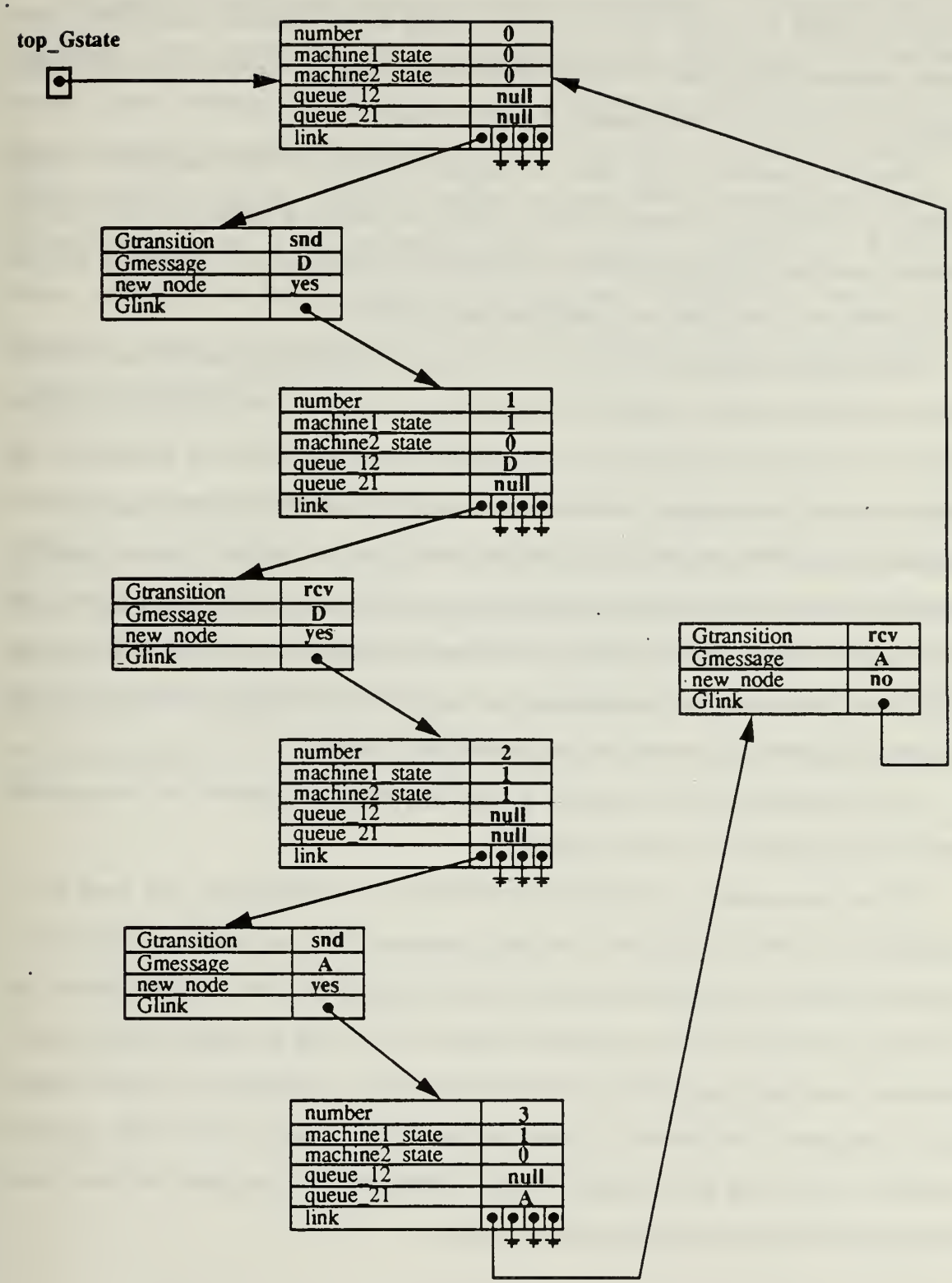


Figure 13: CFSM internal reachability graph, *stop_and_wait*

Note that when a tuple is generated a data structure representing a transition and a node are separately added. The field *new_node* was included in the transition structure to allow proper traversal of the graph. The current version of the program allows for four transitions from each tuple. This can be expanded upward if needed.

Exception handlers were used to maintain control in the reachability graph construction. Whenever a queue or stack is empty the control is handed to the exception handler to continue program execution. The exception handlers allow definition of specific error conditions to be sensed and appropriate action taken.

During graph construction, global state tuples are identified that satisfy the deadlock and unspecified reception properties. If a global state node has only receiving transitions from it and both the queues are empty, a deadlock message is displayed to output. If the global state node has outgoing receive transitions and the head of the respective queue does not match the receive transition (assuming the queue(s) are not empty) then an unspecified reception message is displayed to output. When the construction of the graph is complete the adjacency lists are checked for any unexecuted transitions. The contents of the lists are displayed after the output of the graph is done. Unexecuted transitions are identified by the *execution* field, with a *no* entry. For an example see Figure 15

Upon completion of the reachability graph construction, a pointer to the top global state node is passed to the output procedure.

When constructing a reachability graph there are two factors that need to be considered- run time and the size of the graph generated. As noted earlier, a ceiling can be placed on the size of the graph by the user prior to compilation. Ideally, a specification can be input to the program and an analysis could run for as long as needed (perhaps days); however, most computer systems are limited by storage. The question of storage capacity is left to the user of the program. A determination must be made as to how large a graph to anticipate (worst case is the largest integer represented on the register) and how much storage space can the underlying system provide.

The design of the program addresses the issue of running time. The running time, or complexity, of the reachability analysis is dominated by the algorithm that governs the directed graph traversal. All traversals are done in both models (CFSM and SCM) in a recursive, depth first manner. The complexity, or big O notation, for traversals of a directed graph can easily be defined. Consider a reachability graph $G=(V,E)$ consisting of a set V of vertices(nodes), and a set E of edges(transitions). Each edge corresponds to a pair of distinct vertices in the directed graph. The running time or complexity of such a graph traversal is proven by induction to be $O(|V| + |E|)$. A rigorous proof of the complexity appears in [MANB 89].

D. OUTPUT

The output procedure for the CFSM tool displays the reachability graph and associated messages to both a text file and the default output device. The output procedure has as a parameter a pointer to the top global state (node). From the top node the graph is traversed in a depth first manner and saved to an output medium. The contents of the adjacency list are also displayed to output providing a means to cross check the CFSM construction and identify unexecuted transitions as shown for *stop_and_wait* in Figure 14.

```
                REACHABILITY ANALYSIS of : stop_and_wait

1   [ 0 , E , E , 0 ]    -D  [ 1 , D , E , 0 ]    2
2   [ 1 , D , E , 0 ]    +D  [ 1 , E , E , 1 ]    3
3   [ 1 , E , E , 1 ]    -A  [ 1 , E , A , 0 ]    4
4   [ 1 , E , A , 0 ]    +A  [ 0 , E , E , 0 ]    1
```

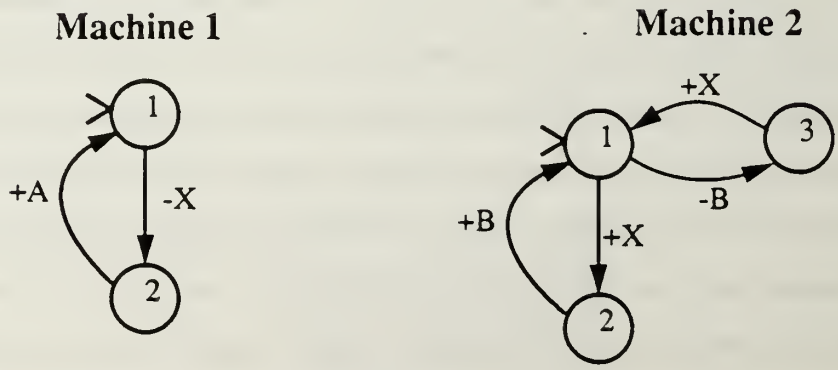
Machine 1 Array Contents			
From	To	Transition	Executed
0	1	snd D	yes
1	0	rcv A	yes

Machine 2 Array Contents			
From	To	Transition	Executed
0	1	rcv D	yes
1	0	snd A	yes

* The nodes generated by the analysis were done in a breadth first manner

Figure 14: CFSM, analysis output, *stop_and_wait*

To illustrate the formatting of the model's output an example is presented. Assume that a CFSM exists reflected by the following specification:



The input file is:

```
start
machine 1
state 1
trans -D 2
state 2
trans +A 1
machine 2
state 1
trans +D 2
state 2
trans -A 1
initial_state 1 1
finish
```

The output file reflecting deadlock, unspecified receptions, and unexecuted transitions is shown in Figure 15.

```
                REACHABILITY ANALYSIS of : deadlock_example

1.  [ 1 , E , E , 1 ]    -X  [ 2 , X , E , 1 ]    2
                          -B  [ 1 , E , B , 3 ]    3
2   [ 2 , X , E , 1 ]    -B  [ 2 , X , B , 3 ]    4
                          +X  [ 2 , E , E , 2 ]    7
7   [ 2 , E , E , 2 ]  ------- ******* DEADLOCK Condition *********
3   [ 1 , E , B , 3 ]    -X  [ 2 , X , B , 3 ]    4
4   [ 2 , X , B , 3 ]    +X  [ 2 , E , B , 1 ]    5
5   [ 2 , E , B , 1 ]    -B  [ 2 , E , BB , 3 ]    6
6   [ 2 , E , BB , 3 ]    ******* Unspecified Reception ******
```

Machine 1 Array Contents			
From	To	Transition	Executed
1	2	snd X	yes
2	1	rcv A	no

Machine 2 Array Contents			
From	To	Transition	Executed
1	3	snd B	yes
1	2	rcv X	yes
2	1	rcv B	no
3	1	rcv X	yes

```
* The nodes generated by the analysis
  were done in a depth first manner
```

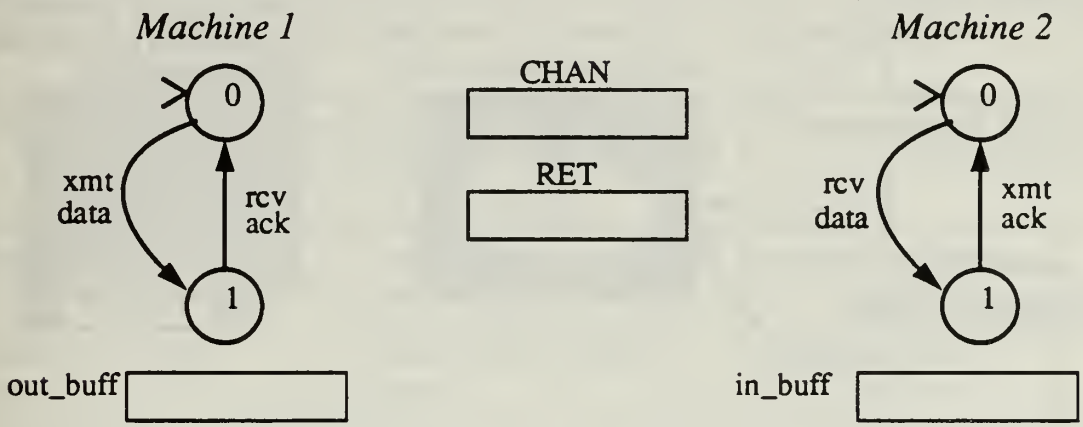
Figure 15: CFSM, analysis output, deadlock/unspecified reception/unexecuted transition example.

IV. AN AUTOMATED TOOL FOR SCM REACHABILITY ANALYSIS

In this chapter, a program is introduced that automates the SCM model . It provides an intuitive environment to input a protocol specification and receive the analysis in an understandable format. Since the model only uses variables and finite state machines to describe a protocol's behavior, it is considered an approximate model. There are certain details of protocol design, such as message and header format, that are abstract from the analysis. The succinctness of a protocol representation helps analyze the logic and structure without getting lost in a myriad of detail.

The organization of this program is similar to that of the CFSM program. A means for input, output, global reachability analysis, and system reachability analysis are highlighted. Excerpts of the code are accompanied by a brief explanation of structure and application.The formal definition of the SCM model found in Chapter II is the basis for constructing the program.

The protocol specification and analysis of the *stop_and_wait* data link protocol will be used throughout this chapter to demonstrate the interface of the program to the user. The specification for the sample protocol is shown in Figure 16.



Transition	Enabling Predicate	Action
xmt_data	*CHAN = E ∧ out_buff /= E*	*CHAN := out_buff*
rcv_ack	*RET = A*	*RET := E* *CHAN := E*
rcv_data	*CHAN /= E*	*in_buff := CHAN*
xmt_ack	*true*	*RET := A* *in_buff := e;*

Figure 16: SCM, specification for *stop_and_wait.*

A. PROGRAM STRUCTURE

The structure of the SCM program is similar to the CFSM implementation. There must be a means for input, output, and reachability analysis. The input is more complex because not only must the FSM's must be entered, but also variable definitions and the associated predicate-action table as shown in Figure 16. The input can be viewed as hierarchical. The global and system reachability analysis are performed using different algorithms and are

described in later sections. The code for producing output is identical to the CFSM program with some addition to allow the user to tailor variable output.

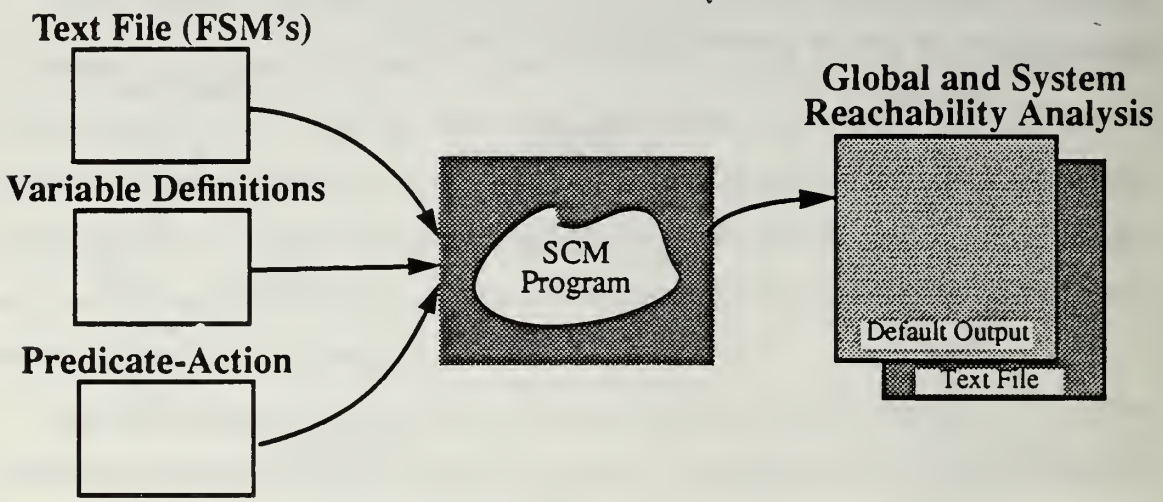


Figure 17: SCM run time behavior.

The program, written in Ada, consists of packages, procedures, and functions that make up the basic structure mentioned above. A package specifies a group of logically related entities, such as types, and objects of those types as defined in [GONZ 91] and [SKAN 88]. The procedures and function that were subject to change/updates were also treated as separate compilation units. To give a 'feel' for the different components of the program, the separate compilation units and the files that contain them are shown in Table 4.

TABLE 4: SCM compilation units.

Compilation Unit	Description	File
read_in_file	parses text input file	input.a
load_machine_array	builds machine adjacency lists form parsed file	input.a
build_Gstate_graph	builds global reachability analysis graph	global_reachability.a

Compilation Unit	Description	File
clear_pointers	clears the values for another input file	global_eachability.a
search_for_Gtuple	performs BFS search of global reachabililty graph	global_search.a
IsEqual	compares global records and associated transitions for equality	global_search.a
build_Sstate_graph	builds system reachability analysis graph	system_reachability.a
search_for_Stuple	perfoms BFS search of graph	system_search.a
IsSysStateEqual	compares system records and associated transitions for equality	system_search.a
output_Gstate_node	format for node output	global_output.a
output_Gstate_transition	format for transition output	global_output.a
output_Gstates	traverses graph and outputs nodes and transitions	global_output.a
output_machine_arrays	format output of contents of adjacency lists	global_output.a
output_Sstate_node	format for node output	system_output.a
output_Sstate_transition	format for transition output	system_output.a
output_Sstates	traverses graph and outputs nodes and transitions	system_output.a
output_Gtuple	format global record for output	user_output.a
variable_definitions	user defined protocol variables	user_definitions.a
Analyze_Predicates	performs analysis of predicates and determines which transitions are enabled	predicate_action.a

Compilation Unit	Description	File
Action	changes the global and local variables based on the transition executed	predicate_action.a

The user has access to the last three files shown in Table 4. The variable *definition* package, *Predicate_Analysis* function, and *Action* procedure contained in these files are modified by the user to reflect the specific protocol to be analyzed. Formats for each unit will be outlined in Sections B, C, and D. The other files and procedures will remain hidden from the user because they are independent of any protocol to be analyzed. Figure 18 shows the files and generic units used at compilation time.

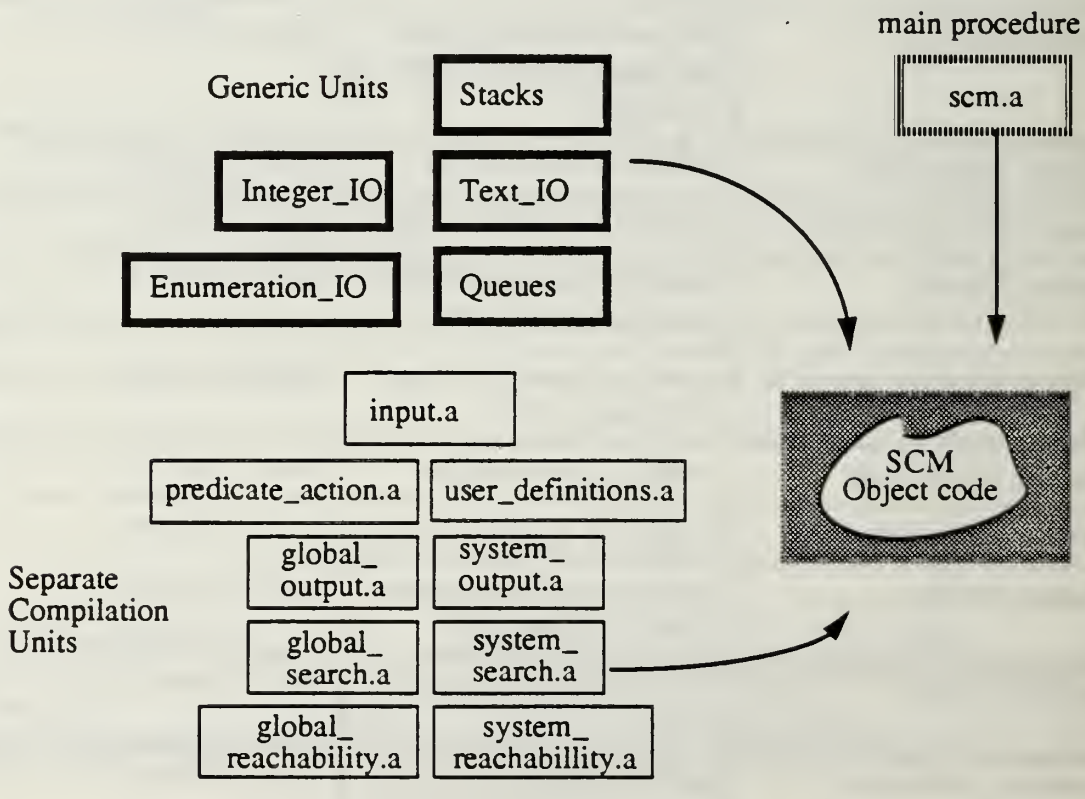


Figure 18: SCM compilation units.

B. INPUT

An intuitive and understandable means to input a specification is helpful in any protocol analysis program. A protocol specification is divided into shared and local variable definitions, predicate-action table representation, and finite state machine storage structure.

The different parts to each specification to be analyzed must be input in a certain order. The *definition* package, *Analyze_Predicate* function, and *Action* procedure must be constructed and compiled before the program is executed. When the program is executed the user then inputs the FSM text file and obtains the reachability analysis. Since the compilation of the program depends on the variables in the *definition* package, this package is written and compiled first. This is a technique to verify variable definition correctness in the Ada environment. Once the *definitions* package is compiled the *Analyze_Predicate* function and *Action* procedure can then be compiled This step-wise refinement facilitates error free specification representation.

At any point in execution of the program the status of all variables is kept in the global state record. Each node in the global reachability graph has a copy of this record. Within the *main* procedure a global state record is declared as

```
type Gstate_record_type is
record
     machine1_state     : machine1_state_type;
     machine2_state     : machine2_state_type;
     global_variables   : global_variable_type;
end record;
```

Having the machine and global types defined in a separate package ensures that only select pieces of code can be modified by the user. The *definitions* package contains the *machine1_state_type*, *machine2_state_type*, and *global_variable_type* declarations; thus this package must be compiled first.

The order that each input category is covered in this chapter reflects order that the protocol specification should be constructed and compiled.

1. Protocol Variable Definitions

The user defines the protocol environment variables in the *definitions* package. Variables can either be local to a specific machine or global to the system. The global variables are considered *shared* and allow communication between the machines in the system. The local variables are only visible to the machine that they are defined for. A discrete variable can be one of the many Ada defined types such as:

```
integer     natural     character
array       digit       boolean
record      access
```

These types variables, or their subtypes, can be used to define protocol environment.

A template for the *definitions* package is illustrated in Figure 19. The shaded areas of the figure are where the variables of the protocol are inserted. All other code should remain unchanged. Additional type declarations should be placed before the machine type

declarations. The *state_number* of each machine is initialized to one even though this maybe different based on the FSM text file is input (the initial state is explicitly given.)

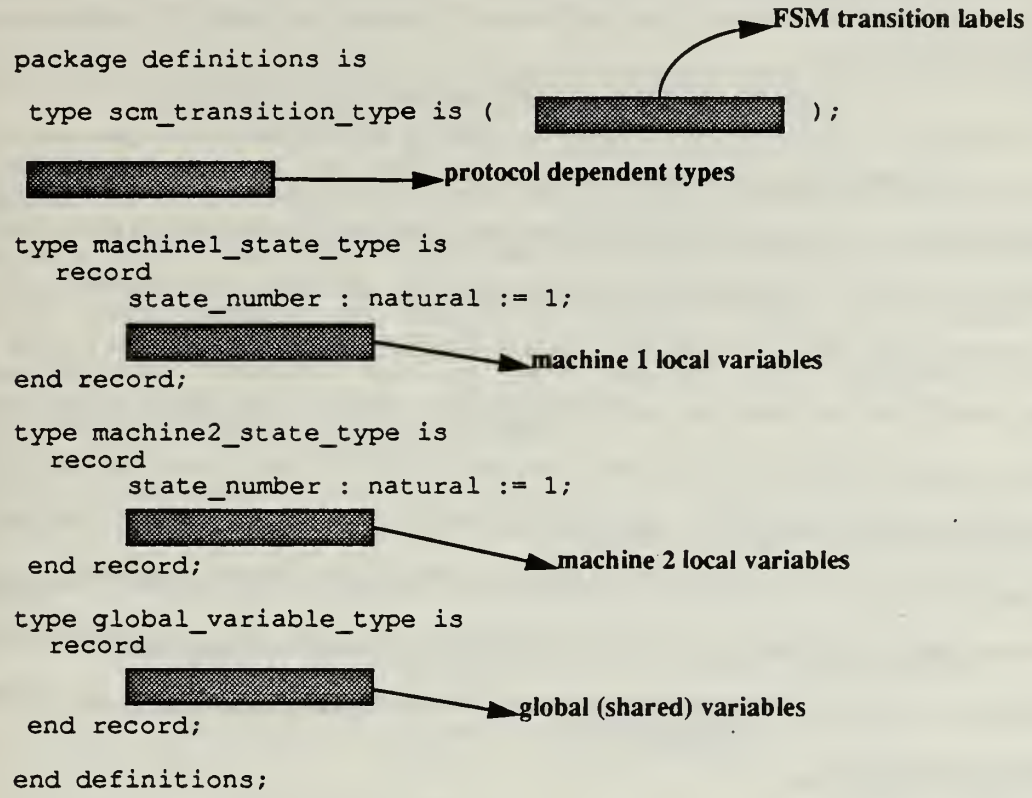


Figure 19: SCM, *definitions* package template.

The variable declarations for the *stop_and_wait* protocol are:shown in Figure 20.

```
package definitions is

        type scm_transition_type is    (snd_data, rcv_data,
                                        snd_ack, rcv_ack);

        type buffer_type is (d,e,a);

        type machine1_state_type is
              record
                    state_number       : natural          := 1;
                    out_buff           : buffer_type      := d;
        end record;

        type machine2_state_type is
              record
                    state_number       : natural          := 1;
                    in_buff            : buffer_type      := e;
        end record;
```

```
        type global_variable_type is
              record
                    CHAN   : buffer_type      := E;
                    RET    : buffer_type      := E;
        end record;

end definitions;
```

Figure 20: SCM, definitions package, *stop_and_wait.*

The transitions are represented as *xmt_data*, *xmt_ack*, *rcv_data*, and *rcv_ack* instead of *-D*, *-A*, *+D*, and *+A*. Machine one has a local variable that serves as an out-bound buffer (*out_buff*). It is initialized with data present in the buffer, represented by '*d*'. The only machine that sees the variable contents is Machine one. Machine two is similar in that it has a buffer for receiving (*in_buff*) data from the channel. The global variables are the shared variables channel (*CHAN*) and a return link (*RET*). Both variables are initialized empty and can be accessed by each machine. The values that *CHAN*, *RET*, *in_buff*, and *out_buff* can have are defined as a *buffer_type*. The *buffer_type* variables can have the values *e* (empty), *d* (data), or *a* (acknowledgement). The *stop_and_wait* protocol example shows how easily variables can be represented. All the text in bold lettering are user defined variables and types.

2. Predicate-Action Table Representation

The predicate-action table serves as the engine to the analysis. The enabling predicate defines the logic that must hold true for the transition to be taken (refer to Table 1). Local and global variables must meet these conditions. A number of transitions could be enabled, but, for a transition to be executed the state of the machine must be considered. The action column of the predicate-action table identifies the variable changes that must take place when the transition is executed. The program captures the essence of the predicate-action table by breaking the components of the table into subprograms. A subprogram in the Ada environment is a function, procedure, or package. Since the user must have access to a number of the subprograms they are represented as separate compilation units.

The first subprogram is the *Analyze_Predicate* function. A function is a subprogram that returns a value to the location in which it was invoked. It can only have input parameters.The function is handed the machine local variables and the system global variables as input parameters. Since more than one transition could be enabled, a stack is used to place all transitions that are enabled. A transition is pushed onto the stack if it is enabled and the function returns the entire *transition_stack*. From the *transition_stack* values and a pointer to the current state in machine adjacency matrix, a determination is made on which transition can actually be executed. There are a number of *Analyze_Predicate* functions, one for each machine. The template for the *Analyze_Predicate* function is shown in Figure 21.

```
separate (main)
function Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type)
        return transition_stack_package.stack is
begin

         MakeEmpty(transition_stack);

        if ( [redacted] ) then
              Push(transition_stack, [redacted] );
        end if;

        return transition_stack;
end Analyze_Predicates_Machine1;
```

enabling condition

enabled transition

Figure 21: SCM, *Analyze_Predicate* function template.

Once a transition is executed, changes must be made to some or all the variables. A procedure using a case statement was the simplest way to make the changes to the global state record. The *Action* procedure has three parameters: the transition that is executed and the current global state record are *in* parameters, and the updated global state record is the *out* parameter. The transition is passed into the procedure and a case statement determines which series of instructions are to be executed. These instructions make the appropriate changes to the protocol environment variables. The *out_system_state* is handed out of the

procedure containing the changed protocol variables. The template for the *Action* procedure is shown in Figure 22.

```
separate (main)
procedure Action(  in_system_state    : in Gstate_record_type;
                   in_transition      : in scm_transition_type;
                   out_system_state   : out Gstate_record_type) is
begin
       case (in_transition) is
             when (                ) =>
                                                  enabled transition
                                                  action taken
             when others =>
                   put_line("Error in the Action procedure");
       end case;
end Action;
```

Figure 22: SCM, *Action* procedure template

The three subprograms that reflect the logic of the predicate-action table are grouped together in one file (predicate_action.a). The file for the *stop_and_wait* protocol is:

```
separate (main)
function Analyze_Predicates_Machine1(local : machine1_state_type;
                                     GLOBAL: global_variable_type)
       return transition_stack_package.stack is
begin
       MakeEmpty(transition_stack);
       if ((local.out_buff /= e) and (GLOBAL.CHAN = E)) then
             Push(transition_stack,xmt_data);
       end if;
       if (GLOBAL.RET = A) then
             Push(transition_stack,rcv_ack);
       end if;
       return transition_stack;
end Analyze_Predicates_Machine1;

separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                     GLOBAL: global_variable_type)
       return transition_stack_package.stack is
begin
       MakeEmpty(transition_stack);
       if ((GLOBAL.CHAN /= E) and (local.in_buff = e)) then
             Push(transition_stack,rcv_data);
       end if;
       Push(transition_stack,xmt_ack);
       return transition_stack;
end Analyze_Predicates_Machine2;separate (main)

separate (main)
```

```
procedure Action(in_system_state      : in Gstate_record_type;
                 in_transition        : in scm_transition_type;
                 out_system_state     : out Gstate_record_type) is
begin
       case (in_transition) is
             when (xmt_data) =>
                   out_system_state.GLOBAL_VARIABLES.CHAN :=
                   in_system_state.machine1_state.out_buff;
             when (rcv_ack) =>
                   out_system_state.GLOBAL_VARIABLES.RET := E;
                   out_system_state.GLOBAL_VARIABLES.CHAN := E;
                   out_system_state.machine2_state.in_buff := e;
             when (xmt_ack) =>
                   out_system_state.GLOBAL_VARIABLES.RET := A;
             when (rcv_data) =>
                   out_system_state.machine2_state.in_buff :=
                   in_system_state.GLOBAL_VARIABLES.CHAN;
              when others =>
                   put_line("Error in the Action procedure");
       end case;
end Action;
```

Figure 23: SCM, analyze_predicate.a, *stop_and_wait.*

The bold text in the code indicates what the user provided as input to define the specification shown in the *stop_and_wait* predicate-action table (See Figure 16.)

3. Finite State Machines

The FSM's are input as a text file during program execution. This file is built by the user with a set of language rules similar to the Backus-Naur Form (BNF) shown in Chapter III. The only change to the format of the input is the transition (`trans`) lines. In the CFSM model only send and receive transitions were allowed; whereas in the SCM model a transition can have any label that follows the enumeration rules. The lines of the text file are buffered and parsed. From the parsed line groups of strings called tokens are manipulated as described in Chapter III. The list of valid instructions for finite state machine input is:

```
start
machine            <natural>
state              <natural>
initial_state      <natural> <natural>
trans              <enumeration_literal> <natural>
finish
```

The tokens are cast into either enumerated types (instructions) or integers (integer variables). The integer variables have been formally defined within the main procedure in Appendix B. The meaning of the instructions are found in Figure 24.

```
start           Serves as a beginning flag for the file.
machine         Defines the current machine.
state           Defines the current state.
initial_state   The intitial/start state for
                machines one and two.
trans           Transition type and next state.
finish          This token serves as an ending flag for
                the file.
```

Figure 24: Input File definitions.

Representation of a finite state machine using the above convention has some inherent constraints. Rules for constructing enumeration literals must be followed. For instance, the list of values in an enumeration literal can only be character literals and cannot contain a digit in the first position. The input file for *stop_and_wait* is shown in Figure 25:

```
start
machine 1
state 0
trans xmt_data 1
state 1
trans rcv_ack 0
machine 2
state 0
trans rcv_data 1
state 1
trans xmt_ack 0
initial_state 0 0
finish
```

Figure 25: SCM, Input File, *stop_and_wait.*

A data structure that contributes to the reachability analysis is the FSM adjacency list. The adjacency list was chosen as the structure to represent the directed graph of the finite state machines.

The actual data structure to build the adjacency list and the defined constraints are:

```
type machine_array_record_type;
type Mlink_type is access machine_array_record_type;
type executed_type is (yes,no);
type machine_array_record_type is
      record
            transition        : scm_transition_type;
```

```
            next_state          : natural;
            executed            : executed_type;
            Mlink               : Mlink_type
end record;

type machine_array_type is array(positive range<>)
      of Mlink_type;
type system_array_type is array(1..2)
      of machine_array_type;
```

The internal representation of the FSM adjacency lists are the same as Figure 12 except the SCM adjacency lists do not have a message field.

C. REACHABILITY ANALYSIS

The process of generating the set of all states reachable from the initial state is called state reachability analysis. During the reachability analysis a check for deadlock, unspecified reception, and unexecuted transitions are done. The reachability analysis of a specific protocol is done in two phases.

The first is to generate a global state reachability graph. This analysis constructs a graph, whose nodes are the reachable global states, and whose arcs indicate the transitions leading from each global state to another. The global state (node) contains the state of each machine and the values of all the variables.

The second phase of the analysis is to generate an separate system state reachability graph from the global state reachability graph. The system reachability graph contains nodes with just the state information of each machine. The rules for the generation of new states will be discussed in Section 2.

1. Global State Analysis

The process of generating the set of all global states reachable from the initial global state is called global state analysis. This analysis produces a graph, whose nodes are the reachable global states, and whose arcs indicate the transitions leading from each global state to another. The global state of a system consists of the system state tuple, plus the values of all variables, both local and shared. The algorithm as it appears in [LUND 91a] is:

(1) Set each machine to its initial state, and all variables to their initial values. The initial set of reachable global states consists of the initial system state and the value of all variables; the initial graph is a single node representing this state.

(2) From the current global state vector and variable values, determine which transitions are enabled. For each of these transitions, determine the global state which results from its execution. If this state (with the same enabled transitions) has already been generated, then draw an arc from the current state to it, labeling the arc with the transition name. Otherwise, add the new global state to the graph, draw an arc from the current state to it, and label the arc with the name of the transition.

(3) For each new state generated in step 2, repeat step 2. Continue until step 2 has been repeated for each global state thus generated, and no more new states are generated.

The algorithm above was modified to make use of the existing data structures introduced in the CFSM program. A psuedo-code algorithm to construct the global reachability graph is:

```
create top_Gstate pointer and initial node in Gstate_graph
main loop
    for machine_index in 1..number_of_machines loop
        transition_stack := Analyze_predicate(machine_index,current_Gstate)
        while transition_stack is not empty loop
            while current row of machine(machine_index) is not null loop
                if current_Gstate.transition = top of transition_stack then
                    perform Action procedure on current_Gstate and place
                        results in a temp_Gstate
                    search Gstate_graph for temp_Gstate
                    if temp_Gstate found then
                        insert temp_Gstate in Gstate_graph
                        Enqueue pointer to location in Gstate_pointer_queue
                    else
                        link current_Gstate to found Gstate
                        set new_node flag to false
                else
                    traverse current row of machine(machine_index)
            end loop --machine row traversal loop
            Pop a transition from the transition_stack
        end loop --transition loop
        if the Gstate_pointer_queue is not empty then
            Dequeue a pointer
            go to the appropriate row of machine(machine_index)
        else
            exit loop
    end loop --machine loop
    exception
        when Gstate_pointer_queue is empty then
            exit main loop
```

```
        when Gstate_pointer_queue is full then
            print error message
            exit main loop
end main_loop
```

The data structures that are used in the program are defined in the specification of the *main* procedure and the *definitions* package. The machine states and variables along with the global variables are defined in the *definitions* package. The remaining data structures which are hidden from the user are:shown in Figure 26

```
--data structures for the global state tuple(node)
type global_state_type;
type Glink_type is access global_state_type;

--transition structure
type Gstate_transition_type is
       record
            Gtransition : scm_transition_type;
            new_node    : boolean := true;
            Glink       : Glink_type;
            executed    : boolean := false;
 end record;

--global tuple structure
type Gstate_record_type is
       record
            machine1_state     : machine1_state_type;
            machine2_state     : machine2_state_type;
            global_variables   : global_variable_type;
 end record;

--Global state node, contains transition, state, and link information
--needed for building the global state graph
type global_state_type is
       record
            node_number        : natural := 0;
            Gtuple             : Gstate_record_type;
            link1              : Gstate_transition_type;
            link2              : Gstate_transition_type;
            link3              : Gstate_transition_type;
            link4              : Gstate_transition_type;
 end record;
```

Figure 26: SCM, global definitions.

The data structure of the global node (*global_state_type*)encapsulates the information contained in the global state record. The global transition record has the type of transition and information about the node it is pointing to. If the node it is pointing to is a newly created node the new field is set to false, otherwise it maintains its initialized value of true. It also has a *visited* field, used during the construction of the system state

reachability graph. The internal representation of the graph generated by the algorithm for *stop_and_wait* protocol highlights all the data structure used (see Figure 27).

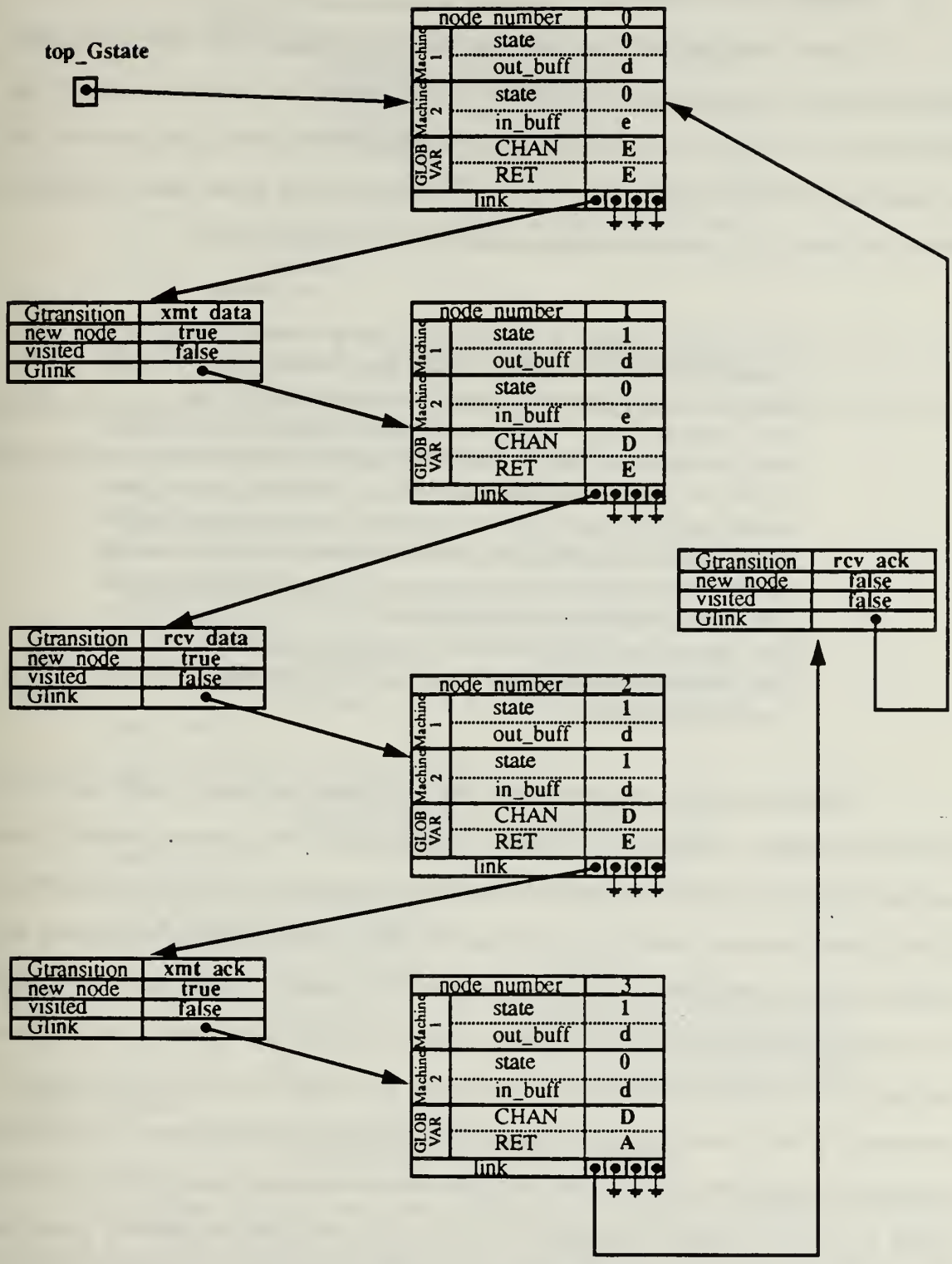


Figure 27: SCM internal global reachability graph, *stop_and_wait.*

2. System State Analysis

System state analysis is similar to global state analysis. The number of states generated by the system state analysis is less than or equal to the number of states in the CFSM model or the global analysis of the SCM model. Only the states of the machines and the transitions from those states are considered when generating a new state. The formal steps in constructing a system state graph as it appears in [LUND 91a] is:

> (1) Set each machine to its initial state, and all variables to their initial values. The initial set of reachable system states consists of only the initial system state; the initial graph is a single node representing this state.
> (2) From the current system state vector and variable values, determine which transitions are enabled. For each of these transitions, determine the system state which results from its execution. If this state (with the same enabled transitions) has already been generated, then draw an arc from the current state to it, labeling the arc with the transition name. Otherwise, add the new system state to the graph, draw an arc from the current state to it, and label the arc with the name of the transition.
> (3) For each new state generated in step 2, repeat step 2. Continue until step 2 has been repeated for each system state thus generated, and no more new states are generated.

The portion of the program that builds the system state graph makes use of the information already available in the global state graph. In the current version of the program the global reachability graph is constructed followed by the system reachability graph. Future versions would allow the user to select which analysis to perform but currently both are constructed and output. The pointer to the initial global state is provided as an input parameter to the *build_Sstate_graph* procedure. The global state graph is traversed in a breadth first manner, as the nodes are visited the system state graph is constructed. If the system state graph were being constructed independently of the global state graph the algorithm would be very similar to the one in the previous section. Since the system state graph is being constructed based on only the global reachability graph the machine matrices are not used. The psuedo-code algorithm for this approach is:

```
create top_Sstate pointer and build initial_Sstate node;
main loop
      while( current_Gstate.link not null and not visited) loop
            visit a Gstate_graph node using BFS traversal
            mark the link taken as visited
            create a temp_Sstate with values of current_Gstate
            search Sstate_graph for temp_Sstate
                  if temp_Sstate found then
                        insert temp_Sstate in Sstate_graph
                  else
                        link current_Sstate to found Sstate
                        set new_node flag to false
      end loop
end main loop
```

The data structures for the system state graph construction, except for the transition labels, are completely hidden from the user. The system related data structures as they appear in the *main* procedure are shown in Figure 28.

```
type system_state_type;
type Syslink_type is access system_state_type;

--transition structure for system state
type Sstate_transition_type is
      record
            Stransition       : scm_transition_type;
            new_node          : boolean := true;
            Syslink           : Syslink_type;
end record;

type Sstate_record_type is
      record
            machine1_state    : natural := 0;
            machine2_state    : natural := 0;
end record;

--system state structure
type system_state_type is
      record
            node_number       : natural := 0;
            Stuple            : Sstate_record_type;
            link1             : Sstate_transition_type;
            link2             : Sstate_transition_type;
            link3             : Sstate_transition_type;
            link4             : Sstate_transition_type;
end record;
```

Figure 28: SCM, system definitions.

To follow through with the *stop_and_wait* analysis example, an internal representation of the system state graph is shown in Figure 29. Although this example does give a graphical picture of how the data structures are used it does not show the advantages

of a system state analysis over global state analysis. Examples covered in Chapter V illustrate how much smaller system state graphs can be when compared to the global analysis graphs.

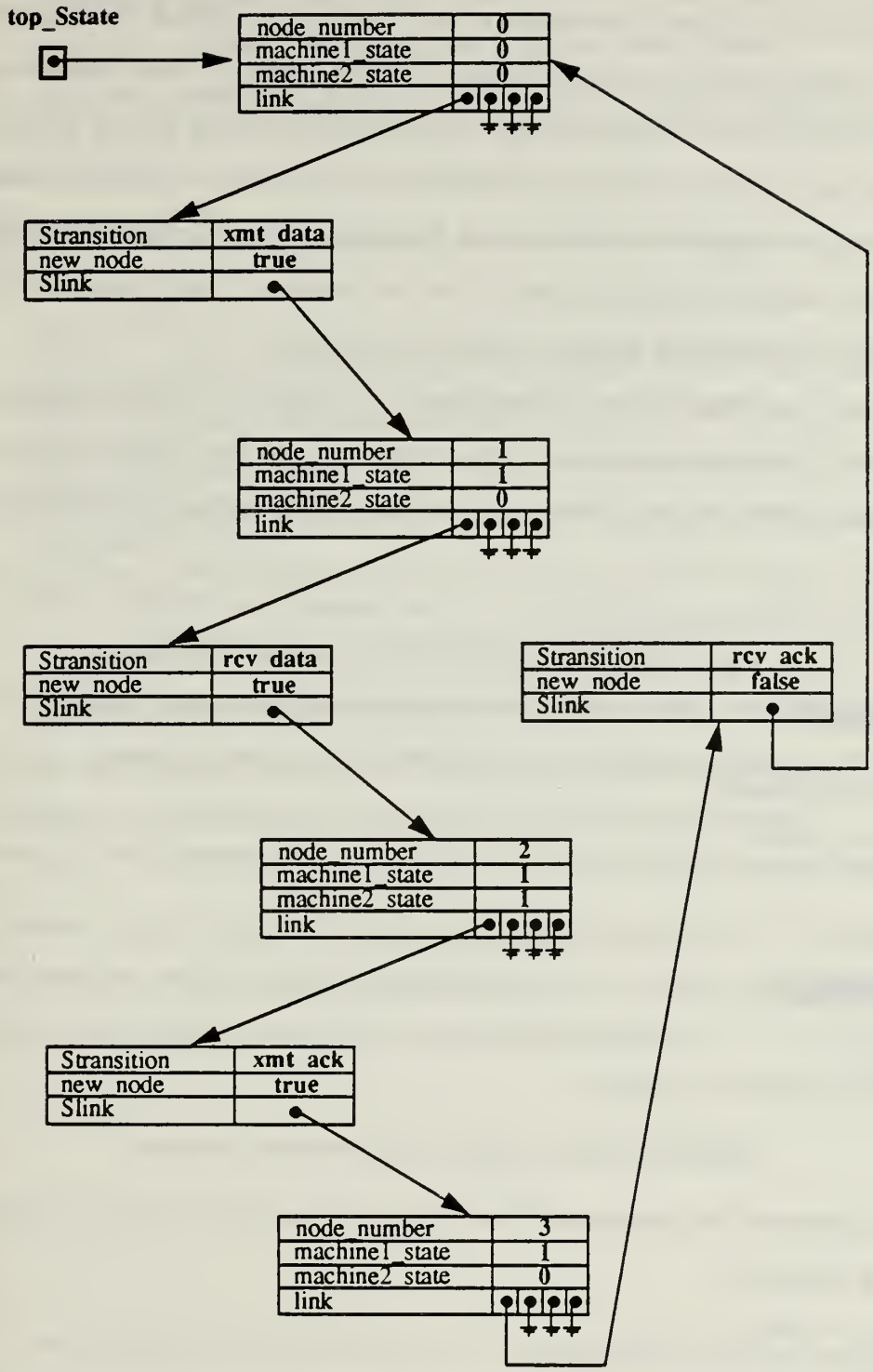


Figure 29: SCM internal system reachability graph, *stop_and_wait.*

D. OUTPUT

Output of the analysis is provided to a text file and to a default device(workstation display). Figure 32 shows a captured image of default output to a workstation screen for *stop_and_wait*. There are features available to allow the user to step through the output one screen at a time. Output messages are provided to the user when a deadlock, unexecuted transition, or an unspecified reception occur. A message is also displayed when the length of the graph exceeds the bounds defined by the user (capacity of the channel is exceeded.) The contents of the machine adjacency lists are also output.

The user may format output for the global state graph. This is done through the file user_ouput.a. The procedure, *output_Gtuple*, contained in the file allows the user to format the variables for default output.The template for output procedures are found in Figure 30.

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
put(" [" & integer'image(tuple.machine1_state.state_number) & " , ");
```

user defined format of variables using text_IO

```
put(" ," & integer'image(tuple.machine2_state.state_number) & " ] ");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type) is
begin

put(reach," [" & integer'image(tuple.machine1_state.state_number));
```

user defined format of variables using text_IO

```
put(reach," ," & integer'image(tuple.machine2_state.state_number) &
" ] ");
end output_Gtuple_to_file;
```

Figure 30: SCM, *output_Gtuple* procedure template.

An example of how a user could format output is given for the stop_and_wait protocol is given in Figure 31

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
        put(" [" & integer'image(tuple.machine1_state.state_number)
        & " , ");
        put(tuple.machine1_state.out_buff, set => lower_case);
```

```
        put(" , ");
        put(tuple.GLOBAL_VARIABLES.CHAN, set => upper_case);
        put(" , ");
        put(tuple.GLOBAL_VARIABLES.RET, set => upper_case);
        put(" , ");
        put(tuple.machine2_state.in_buff, set => lower_case);
        put(" ," & integer'image(tuple.machine2_state.state_number)
        & " ] ");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type) is
begin

        put(reach," [" &
        integer'image(tuple.machine1_state.state_number) & " , ");
        put(reach,tuple.machine1_state.out_buff, set => lower_case);
        put(reach," , ");
        put(reach,tuple.GLOBAL_VARIABLES.CHAN, set => upper_case);
        put(reach," , ");
        put(reach,tuple.GLOBAL_VARIABLES.RET, set => upper_case);
        put(reach," , ");
        put(reach,tuple.machine2_state.in_buff, set => lower_case);
        put(reach," ," &
        integer'image(tuple.machine2_state.state_number) & " ] ");
end output_Gtuple_to_file;
```

Figure 31: SCM, output format, *stop_and_wait.*

Consistent with previous examples the boldface code is that which the user provides. The user does not provide any parameters for system state output. The output shown in Figure 32 was formatted according to the procedures used above.

```
REACHABILITY ANALYSIS of : stop_and_wait

Global State GRAPH

0   [ 0 , d , E , E , e , 0 ]   xmt_data   1
1   [ 1 , d , D , E , e , 0 ]   rcv_data   2
2   [ 1 , d , D , E , d , 1 ]   xmt_ack    3
3   [ 1 , d , D , A , d , 0 ]   rcv_ack    0

System State GRAPH

0   [ 0 , 0 ]    xmt_data   [ 1 , 0 ]  1
1   [ 1 , 0 ]    rcv_data   [ 1 , 1 ]  2
2   [ 1 , 1 ]    xmt_ack    [ 1 , 0 ]  3
3   [ 1 , 0 ]    rcv_ack    [ 0 , 0 ]  0
```

Machine 1 Array Contents			
From	To	Transition	Executed
0	1	xmt_data	yes
1	0	rcv_ack	yes

Machine 2 Array Contents			
From	To	Transition	Executed
0	1	rcv_data	yes
1	0	xmt_ack	yes

Figure 32: SCM, analysis output, *stop_and_wait*.

V. AN AUTOMATED ANALYSIS OF SELECTED DATA LINK PROTOCOLS

In this section the programs that were developed in the last two chapters will be demonstrated. Some well known data link protocols will be analyzed using the CFSM and SCM programs. The CFSM program will be used to analyze the alternating bit and the sliding window protocols. The SCM program will be used to analyze go_back_n and selective repeat protocols. In each analysis the specification will be described; the program input and results are in the Appendices of this publication.

A. CFSM MODEL

The examples used as input show the advantages of the CFSM program. The alternating bit protocol analysis was chosen as a simple class of protocols. The sliding window with a window size of three shows how a graphically complex protocol can be analyzed quite easily.

1. Alternating Bit Protocol

The specification of the alternating bit protocol will be used as the first example for the CFSM program. The protocol consists of two machines. Machine one serves as a sender and Machine two as the receiver. The sender sends a message(-X) to the receiver. The receiver then accepts the message (+X) and sends an acknowledgment (-A). The acknowledgment at the machine level is done with the toggling of a bit, wherein the name

alternating bit is derived. The sender is clear to send another message when the acknowledgment is received.

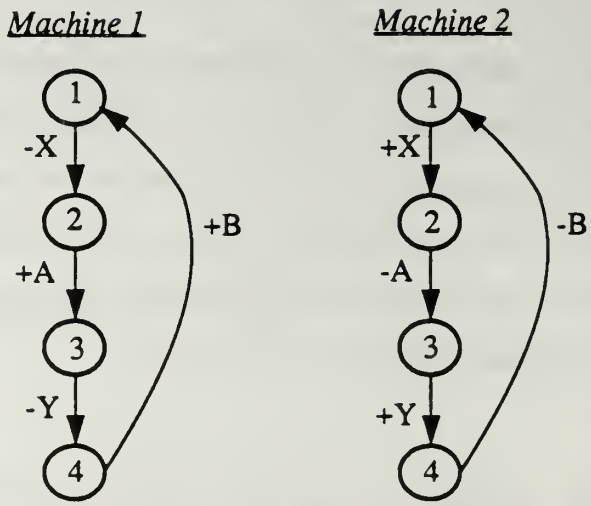


Figure 33: CFSM specification, *Alternating Bit.*

The input file for the specification is

```
start
machine 1
state 1
trans -X 2
state 2
trans +A 3
state 3
trans -Y 4
state 4
trans +B 1
machine 2
state 1
trans +X 2
state 2
trans -A 3
state 3
trans +Y 4
state 4
trans -B 1
initial_state 1 1
finish
```

The analysis of the alternating bit specification is:

```
REACHABILITY ANALYSIS of : output.alt_bit

1   [ 1 , E , E , 1 ]   -X  [ 2 , X , E , 1 ]    2
2   [ 2 , X , E , 1 ]   +X  [ 2 , E , E , 2 ]    3
3   [ 2 , E , E , 2 ]   -A  [ 2 , E , A , 3 ]    4
4   [ 2 , E , A , 3 ]   +A  [ 3 , E , E , 3 ]    5
5   [ 3 , E , E , 3 ]   -Y  [ 4 , Y , E , 3 ]    6
6   [ 4 , Y , E , 3 ]   +Y  [ 4 , E , E , 4 ]    7
7   [ 4 , E , E , 4 ]   -B  [ 4 , E , B , 1 ]    8
8   [ 4 , E , B , 1 ]   +B  [ 1 , E , E , 1 ]    1
```

Machine 1 Array Contents			
From	To	Transition	Executed
1	2	snd X	yes
2	3	rcv A	yes
3	4	snd Y	yes
4	1	rcv B	yes

Machine 2 Array Contents			
From	To	Transition	Executed
1	2	rcv X	yes
2	3	snd A	yes
3	4	rcv Y	yes
4	1	snd B	yes

* The nodes generated by the analysis were done in a breadth first manner

2. A More Complex Example: The Sliding Window Protocol.

The analysis of a sliding window protocol is a more complex example. To represent the protocol as a set of graphical finite state machines can be quite tedious. The essence of the protocol must be captured with the use of transitions, oftentimes this can lead to an intricate diagram as in this example.

The sliding window protocol can also be represented as a two machine CFSM. As in the previous example, Machine one is the sender and Machine two is the receiver. At any instant of time the sender maintains a list of consecutive sequence numbers corresponding to frames it is permitted to send[TANE 81] These frames are said to fall

within the sending window. The receiver also maintains a receiving window corresponding to frames it is permitted to accept. The sending window and the receiving window need not have the same lower and upper limits, or even have the same size.

A window size of three is used in the specification given in Figure 34. The messages or packets are shown as transitions labeled X, Y, and Z and the acknowledgments are A, B, and C.

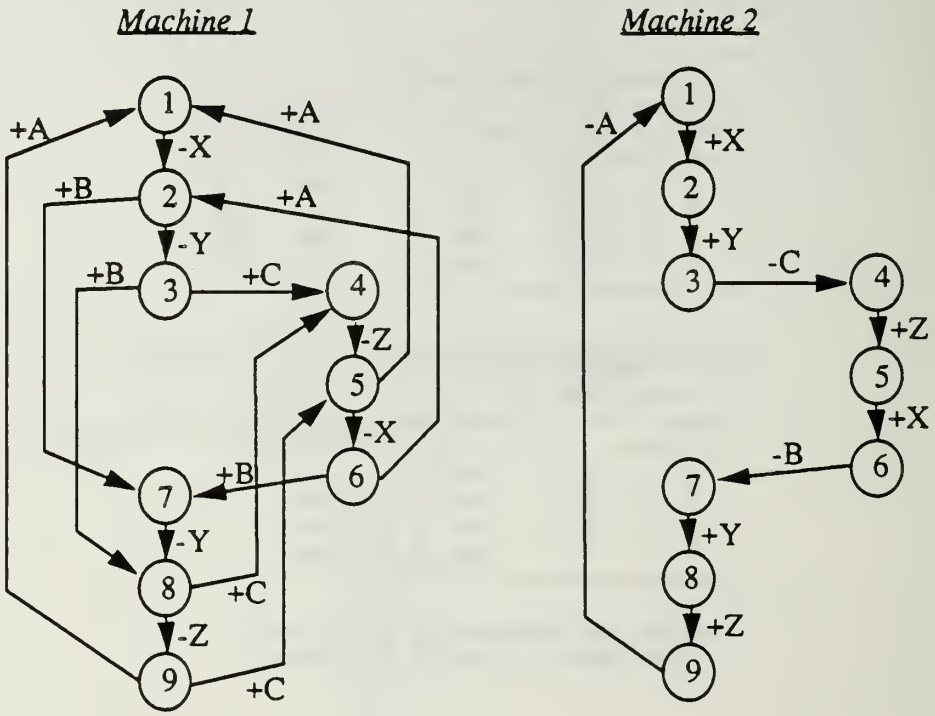


Figure 34: CFSM specification, *sliding window (w=3).*

The text file of the specification is:

```
start
machine 1
state 1
trans -X 2
state 2
trans +B 7
trans -Y 3
state 3
trans +B 8
trans +C 4
state 4
trans -Z 5
state 5
trans -X 6
trans +A 1
state 6
trans +A 2
trans +B 7
state 7
trans -Y 8
state 8
trans -Z 9
state 9
trans +C 5
trans +A 1

machine 2
state 1
trans +X 2
state 2
trans +Y 3
state 3
trans -C 4
state 4
trans +Z 5
state 5
trans +X 6
state 6
trans -B 7
state 7
trans +Y 8
state 8
trans +Z 9
state 9
trans -A 1
initial_state 1 1
finish
```

The analysis of the specification as contained in the output text file:

```
REACHABILITY ANALYSIS of : output_sliding

 1  [ 1 , E , E , 1 ]   -X  [ 2 , X , E , 1 ]    2
 2  [ 2 , X , E , 1 ]   -Y  [ 3 , XY , E , 1 ]   3
                        +X  [ 2 , E , E , 2 ]    4
 3  [ 3 , XY , E , 1 ]  +X  [ 3 , Y , E , 2 ]    5
 4  [ 2 , E , E , 2 ]   -Y  [ 3 , Y , E , 2 ]    5
 5  [ 3 , Y , E , 2 ]   +Y  [ 3 , E , E , 3 ]    6
 6  [ 3 , E , E , 3 ]   -C  [ 3 , E , C , 4 ]    7
 7  [ 3 , E , C , 4 ]   +C  [ 4 , E , E , 4 ]    8
 8  [ 4 , E , E , 4 ]   -Z  [ 5 , Z , E , 4 ]    9
 9  [ 5 , Z , E , 4 ]   -X  [ 6 , ZX , E , 4 ]   10
                        +Z  [ 5 , E , E , 5 ]    11
10  [ 6 , ZX , E , 4 ]  +Z  [ 6 , X , E , 5 ]    12
11  [ 5 , E , E , 5 ]   -X  [ 6 , X , E , 5 ]    12
12  [ 6 , X , E , 5 ]   +X  [ 6 , E , E , 6 ]    13
13  [ 6 , E , E , 6 ]   -B  [ 6 , E , B , 7 ]    14
14  [ 6 , E , B , 7 ]   +B  [ 7 , E , E , 7 ]    15
15  [ 7 , E , E , 7 ]   -Y  [ 8 , Y , E , 7 ]    16
16  [ 8 , Y , E , 7 ]   -Z  [ 9 , YZ , E , 7 ]   17
                        +Y  [ 8 , E , E , 8 ]    18
17  [ 9 , YZ , E , 7 ]  +Y  [ 9 , Z , E , 8 ]    19
18  [ 8 , E , E , 8 ]   -Z  [ 9 , Z , E , 8 ]    19
19  [ 9 , Z , E , 8 ]   +Z  [ 9 , E , E , 9 ]    20
20  [ 9 , E , E , 9 ]   -A  [ 9 , E , A , 1 ]    21
21  [ 9 , E , A , 1 ]   +A  [ 1 , E , E , 1 ]    1
```

Machine 1 Array Contents

From	To	Transition	Executed
1	2	snd X	yes
2	7	rcv B	no
2	3	snd Y	yes
3	8	rcv B	no
3	4	rcv C	yes
4	5	snd Z	yes
5	6	snd X	yes
5	1	rcv A	no
6	2	rcv A	no
6	7	rcv B	yes
7	8	snd Y	yes
8	9	snd Z	yes
9	5	rcv C	no
9	1	rcv A	yes

Machine 2 Array Contents

From	To	Transition	Executed
1	2	rcv X	yes
2	3	rcv Y	yes
3	4	snd C	yes
4	5	rcv Z	yes
5	6	rcv X	yes
6	7	snd B	yes
7	8	rcv Y	yes
8	9	rcv Z	yes
9	1	snd A	yes

The unexecuted transitions account for error control. Error control refers to mechanisms to detect and/or correct errors that occur in the transmission of information. So the unexecuted transitions identified in Machine 2's array are transitions that would have been executed if a loss of a message had occurred. Error and time-out transitions are not shown in the CFSM.

B. SCM MODEL

The examples used as input validate the use of the SCM program as a tool to verify protocols. The *go_back_n* protocol was analyzed first due to the availability of prior modelling done using SCM. The output of the program were compared to the manual SCM modelling results of this protocol. The *selective repeat* specification provided additional evidence as to the programs validity as well as demonstrating how it can be used to improve a specification. In both examples it must be shown that the use of the analysis should help the designer or reviewer to gain a greater understanding of the protocol, as well as in detecting errors.

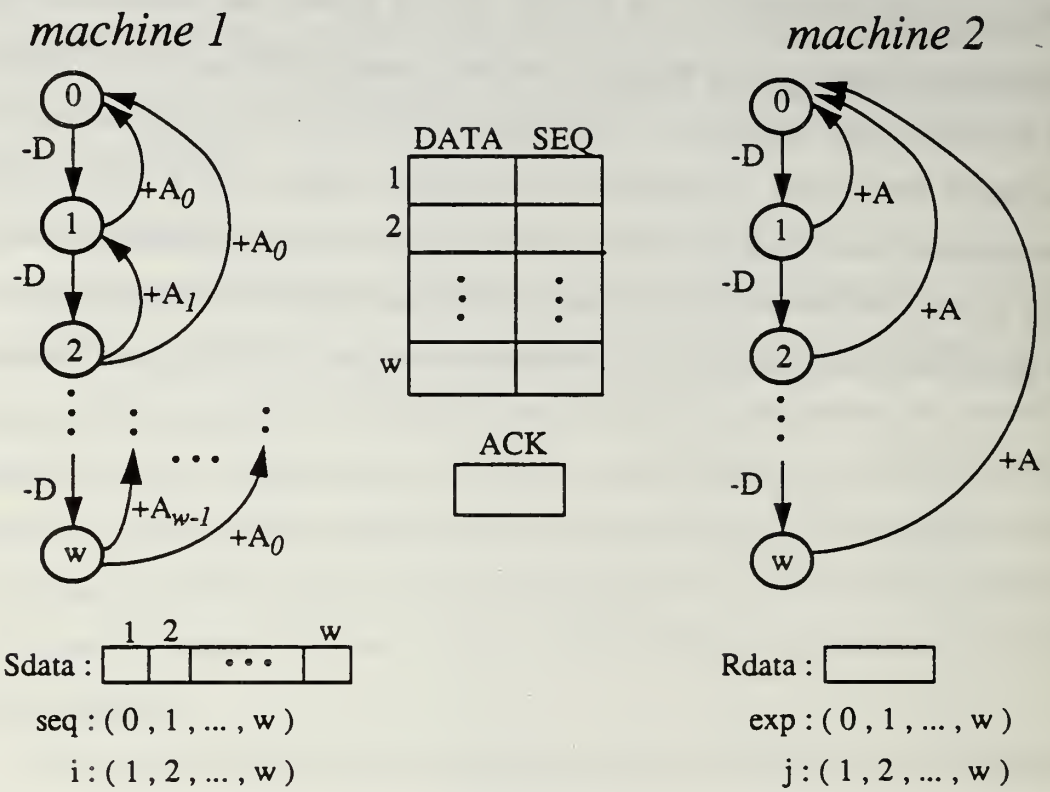
The analysis of any protocol using this program contains varying information. A global reachability graph and system reachability graph are provided. Following the graph is a description of the contents of each machine array upon termination of the graph construction. Error messages (deadlock, unspecified reception, and unexecuted transitions) are placed at the point in the analysis where they occur. The system state graph will be used in this section to provide a means to validate output results. The system state graph can be viewed as a three dimensional object whose tuple values provide a vector to 3 dimensional space.

1. Go Back N

The first protocol which was chosen to model is a go_back_n protocol with a variable window size, which is a subset of the High-level Data Link Control (HDLC) class of protocols. There are two machines in the system, a sender(m_1) and a receiver (m_2). The sender sends data blocks to the receiver, which are numbered sequentially, 0, 1,..., w, 0, 1,...

for a window size of w. The maximum number of data blocks which can be sent without receiving an acknowledgment is w, the window size.The receiver, m_2, receives the data blocks and acknowledges them by sending the sequence number of the next block expected (which is stored in local variable *exp*). The shared variables DATA and SEQ are used to pass messages from sender to receiver, and the shared variable ACK is used to pass acknowledgments back to the *sender*. The *receiver* may acknowledge any number of blocks received up to the window size. Upon receiving the acknowledgment, the *sender* must be able to deduce how many data blocks are being acknowledged. This is done by observing

the difference between the values of the received acknowledgment and the sequence number of the last data block sent.



Transition	Enabling Predicate	Action
-D	*DATA(i)=E ∧ SEQ(i)=E*	*DATA(i) := Sdata(i)* *SEQ(i) := seq* *inc(i,seq)*
$+A_k$ $(0 \le k < w)$	*ACK ⊕ k = seq ∧* *ACK ≠ E* *(next state : k)*	*ACK := E*
+D	*DATA (j) ≠ E ∧ SEQ(j)=exp*	*Rdata := DATA(j)* *DATA(j), SEQ(j) := E* *inc(j,exp)*
-A	*DATA(j)=E*	*ACK := exp*

Figure 35: SCM specification, *Go Back N*, window size of 1..w.

The general specification of the protocol is given in Figure 35. Included in this figure are the state machine diagrams, variables and the predicate action table. Initially, both *sender* and *receiver* are in state 0, arrays DATA and SEQ are empty, and ACK is empty. The domains of DATA, Rdata and Sdata are not specified; these are used to hold user data blocks. Sdata and Rdata are the interface or access points of the higher layer (user) protocol. The local variables for the sender are Sdata, used to store data blocks, *seq*, used to store the sequence number of the next data block to be sent out, and *i*, used as an index into the DATA and SEQ arrays. Initially *seq* is set to 0, and *i* is set to 1. The local variables of the receiver are Rdata, *exp*, and *j*. Rdata is used to receive and store incoming data blocks, *exp* to hold the expected sequence number of the next incoming data block, and *j* is an index into the shared arrays DATA and SEQ.

There are four basic types of transitions described in the predicate action table. In the sender the *-D* transition transmits a data block by placing it into the shared variable DATA(i), and the sequence number into SEQ(i). The send is enabled whenever those variables are empty. The receive transition in the receiver, m_2, is enabled whenever a data block of the appropriate sequence number is in the jth element of DATA and SEQ. An acknowledgment may be sent by m_2 in any state except 0, in which case no unacknowledged data blocks have been received. The $+A$ is a receive transition. If m_1 is in state u, $1 \le u \le w$, and there is a nonempty value in shared variable ACK, then exactly one of the transitions $+A_0$, $+A_1$,..., $+A_{w-1}$ will be enabled; it will be that A_k such that the predicate $ACK \oplus k = seq$ is true, and the next state is k. In the state diagram, all of the transition $+A_k$ are shown using the same vertical line.

a. Input of variable definitions.

A sample interaction using the program for the analysis of go_back_n, w=1,consists of input files and an output file. The variable definitions contained in the user specification file are:

```
package definitions is

        type scm_transition_type is (snd_data, rcv_data,
                                     snd_ack, rcv_ack0, unused);
```

```
        type buffer_type is (d,e,a);
        type buffer_array_type is array(1..1) of buffer_type;
        type seq_array_type is array(1..1) of integer range -1..1;

        type machine1_state_type is
              record
                    state_number : natural := 1;
                    Sdata : buffer_array_type := (others=>d);
                    seq : integer range 0..1 := 0;
                    i : integer range 1..1 := 1;
        end record;

        type machine2_state_type is
               record
                    state_number : natural := 1;
                    Rdata : buffer_type := e;
                    exp : integer range 0..1 := 0;
                    j : integer range 1..1 := 1;
        end record;

        type global_variable_type is
        record
              DATA : buffer_array_type := (others=>e);
              SEQ : seq_array_type := (others=>-1);
              ACK : integer range -1..1 := -1;
        end record;

end definitions;
```

b. Input of predicate analysis.

The analyze predicate functions e contained in the predicate_action.a file are:

```
function Analyze_Predicates_Machine1(local : machine1_state_type;
                                     GLOBAL: global_variable_type)
return transition_stack_package.stack is
        temp1 : integer := GLOBAL.ACK + 0;
begin
        MakeEmpty(transition_stack);
        if ((GLOBAL.DATA(local.i) = E)
        and (GLOBAL.SEQ(local.i) = -1)) then
              Push(transition_stack,snd_data);
        end if;
        if ((temp1 = local.seq) and (GLOBAL.ACK /= -1)) then
              Push(transition_stack,rcv_ack0);
        end if;
        return transition_stack;
end Analyze_Predicates_Machine1;

function Analyze_Predicates_Machine2(local : machine2_state_type;
                                     GLOBAL: global_variable_type)
return transition_stack_package.stack is
begin
        MakeEmpty(transition_stack);
        if ((GLOBAL.DATA(local.j)/=E)
        and (GLOBAL.SEQ(local.j) = local.exp)) then
              Push(transition_stack,rcv_data);
        end if;
        if (GLOBAL.DATA(local.j)=E) then
              Push(transition_stack,snd_ack);
```

```
        end if;
        return transition_stack;
end Analyze_Predicates_Machine2;
```

c. *Input of action table.*

The action procedure is also a separate compilation unit contained in the predicate_action.a file:

```
procedure Action(in_system_state : in out Gstate_record_type;
                 in_transition : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

        temp : integer := 0;
begin
        case (in_transition) is
              when (snd_data) =>
                    out_system_state.GLOBAL_VARIABLES.
                    DATA(in_system_state.machine1_state.i) :=
                          in_system_state.machine1_state.
                          Sdata(in_system_state.machine1_state.i);
                    out_system_state.GLOBAL_VARIABLES.
                          SEQ(in_system_state.machine1_state.i) :=
                          in_system_state.machine1_state.seq;
                    out_system_state.machine1_state.i :=
                          (((in_system_state.machine1_state.i) +
                          1)mod 1) + 1;
                    out_system_state.machine1_state.seq :=
                          (((in_system_state.machine1_state.seq)
                           + 1)mod 2);
              when (rcv_ack0) =>
                    out_system_state.GLOBAL_VARIABLES.ACK := -1;
              when (snd_ack) =>
                    out_system_state.GLOBAL_VARIABLES.ACK :=
                          in_system_state.machine2_state.exp;
                    out_system_state.machine2_state.Rdata := e;
              when (rcv_data) =>
                    out_system_state.machine2_state.Rdata :=
                          in_system_state.GLOBAL_VARIABLES.DATA
                          (in_system_state.machine2_state.j);
                    out_system_state.GLOBAL_VARIABLES.DATA
                          (in_system_state.machine2_state.j) := E;
                    out_system_state.GLOBAL_VARIABLES.SEQ
                          (in_system_state.machine2_state.j) := -1;
                    out_system_state.machine2_state.j :=
                          (((in_system_state.machine2_state.j)
                          + 1)mod 1) + 1;
                    out_system_state.machine2_state.exp :=
                          (((in_system_state.machine2_state.exp)
                           + 1)mod 2);
              when others =>
                    put_line("There is an error in the
                    Action procedure");
        end case;
end Action;
```

d. Input of finite state machines.

And, finally, the input file for the finite state machines is:

```
start
machine 1
state 0
trans snd_data 1
state 1
trans rcv_ack0 0
machine 2
state 0
trans rcv_data 1
state 1
trans snd_ack 0
initial_state 0 0
finish
```

e. Output of analysis.

The output of the analysis is:

```
                REACHABILITY ANALYSIS of : go_back_n_w1

                    Global State GRAPH

0  [ 0 , 0 , 0 , 1 , 0 , 1 , E , -1 , -1 ]     snd_data   1
1  [ 1 , 0 , 1 , 1 , 0 , 1 , D ,  0 , -1 ]     rcv_data   2
2  [ 1 , 1 , 1 , 1 , 1 , 1 , E , -1 , -1 ]     snd_ack    3
3  [ 1 , 0 , 1 , 1 , 1 , 1 , E , -1 ,  1 ]     rcv_ack0   4
4  [ 0 , 0 , 1 , 1 , 1 , 1 , E , -1 , -1 ]     snd_data   5
5  [ 1 , 0 , 0 , 1 , 1 , 1 , D ,  1 , -1 ]     rcv_data   6
6  [ 1 , 1 , 0 , 1 , 0 , 1 , E , -1 , -1 ]     snd_ack    7
7  [ 1 , 0 , 0 , 1 , 0 , 1 , E , -1 ,  0 ]     rcv_ack0   0

                    System State GRAPH

0   [ 0 , 0 ]   snd_data   [ 1 , 0 ]  1
1   [ 1 , 0 ]   rcv_data   [ 1 , 1 ]  2
2   [ 1 , 1 ]   snd_ack    [ 1 , 0 ]  3
3   [ 1 , 0 ]   rcv_ack0   [ 0 , 0 ]  0

-------------------------------------
|      Machine 1 Array Contents      |
-------------------------------------
| From | To | Transition | Executed |
-------------------------------------
|  0   |  1 | snd_data   |   yes    |
|  1   |  0 | rcv_ack0   |   yes    |
-------------------------------------

-------------------------------------
|      Machine 2 Array Contents      |
-------------------------------------
| From | To | Transition | Executed |
-------------------------------------
|  0   |  1 | rcv_data   |   yes    |
|  1   |  0 | snd_ack    |   yes    |
-------------------------------------
```

The output indicates that no errors were encountered. The format for the variables are user dependent and, for brevity sake, the code for formatting the output was not included.

f. System state analysis.

System state analysis is similar to the reachability analysis used with the pure finite state machine model, but the total number of states which must be generated with system state analysis is significantly smaller.

The system state analysis for a window size of 1 is shown in Figure 36.The subscripts are used so that distinct system states having the same tuple may easily be distinguished. The convention is that the subscript is initially 0, and is increased whenever a "-A" transition is taken, by the number of messages which are being acknowledged.

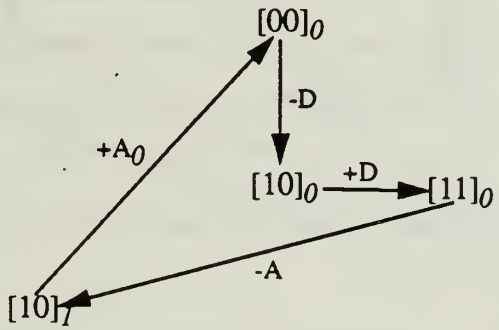


Figure 36: SCM, system state analysis, *Go Back N*, w=1.

The analysis for w=2 is shown in Figure 37. The initial states and variable values are the same as for the w=1, however there are clearly more states in the analysis.

In a comparison between window sizes of 1 and 2, it is noted that the smaller graph is a subgraph of the larger; either can be obtained from the other. If the subscripts are taken as the third coordinate

in a 3-dimensional cartesian coordinate system, with the states of each machine as the first two coordinates. The graph then is the shape of a tetrahedron, with edges which are directed and labeled.

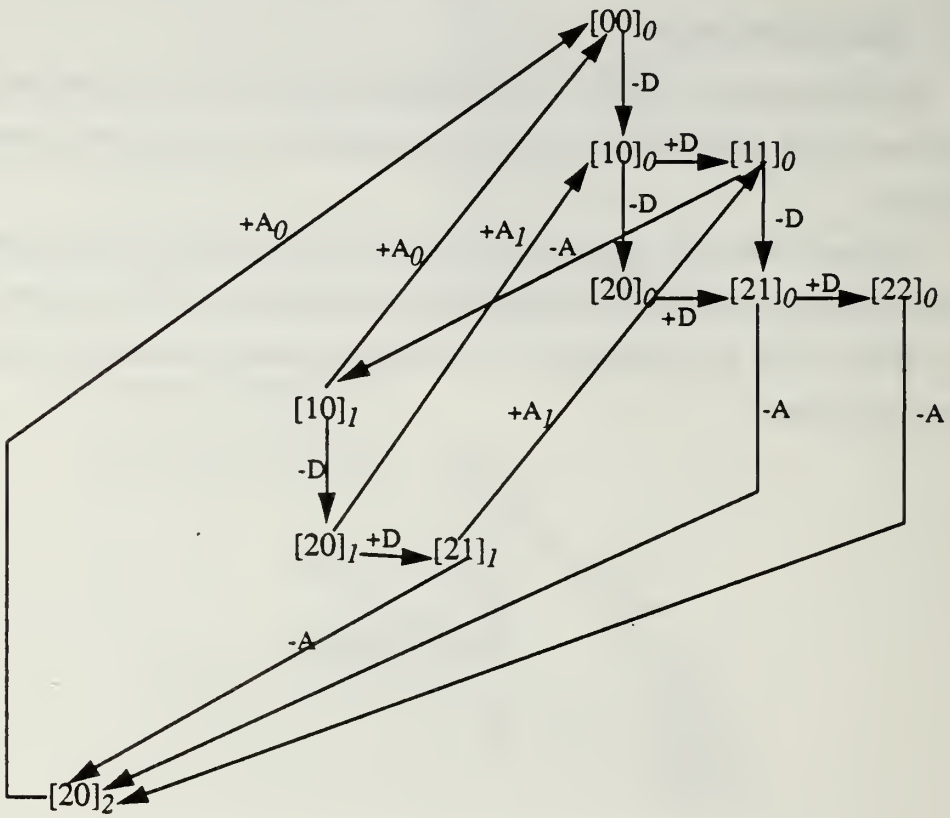


Figure 37: SCM, system state analysis, *Go Back N*, w=2.

The graphs contained in Figure 36 and Figure 37 are defined with respect to a window size *w*. The graphs, DT1(w) for a nonnegative integer w is a labeled, directed graph, defined by the tuple (N, E, L, Φ), where $N = \{ (x, y, z) \mid (0 \le z \le w, z \le x \le w, 0 \le y \le x - z) \}$ is a finite set of nodes, where each node is specified by an ordered triple; $L=\{-D, +D, -A, +A_0, +A_1, ..., +A_{w-1}\}$ is a finite set of label; the set E of edges is a set of ordered pairs $((x_1, y_1, z_1), (x_1, y_2, z_2))$ of nodes from N, and is the union of the following four sets:

$E_1 = \{((x, y, z), (x+1, y, z)) / (x, y, z) \in N, x < w)$

$E_2 = \{((x, y, z), (x, y+1, z)) / (x, y, z) \in N, y < x - z)$

$E_3 = \{((x, y, z), (x, 0, y+z)) / (x, y, z) \in N, (y = x - z), x > z)$

$E_4 = \{((x, y, z), (x-z, y, 0)) / (x, y, z) \in N, z > 0)$

and the mapping $\Phi (L \leftarrow E)$ is defined as follows:

$\forall (x, y, z) \in E_1, \Phi(x, y, z) = -D$

$\forall (x, y, z) \in E_2, \Phi(x, y, z) = +D$

$\forall (x, y, z) \in E_3, \Phi(x, y, z) = -A$

$\forall (x, y, z) \in E_4, \Phi(x, y, z) = +A_k$, where k=x-z

Each node of the graph can be thought of as a point in 3-dimensional space, with nonnegative, integral coordinates (x,y,z). The structure of the graph is a sequence of $w+1$ triangles, one on top of the other, with the largest triangle at the bottom and the smallest is a single point at the top level.

One of the nice features of the geometric structure of this graph is that the state of the system can be easily inferred from the x, y, z coordinates. For example, in Figure 37, point (2,1,0), or system state $[2,1]_0$, the sender has transmitted 2 data blocks for which no acknowledgment has yet been received, the receiver has received 3 of these, but acknowledged non.

Let f(w) be the amount of nodes in a system state graph and g(w) be the amount of nodes in the global reachability graph. The equations for g(w) and f(w), and the lemmas that support them, for the go_back_n protocol are found in [LUND 91a]. For instance, the graph DT1(w) has $f(w) = \frac{1}{6}w^3 + w^2 + \frac{11}{6}w + 1$. The size of the graphs according to window size are:

w	f(w)	g(w)
0	1	1
1	4	8
2	10	60
3	20	240
4	35	700
5	56	1680
5	84	3528
7	120	6720

The output of the program for this protocol was compared to values for f(w) and g(w). Test runs were done for window sizes of 0 thru 5 and the amount of nodes in each graph were consistent with the table above. The specification input files and the output is in Appendices E through H.

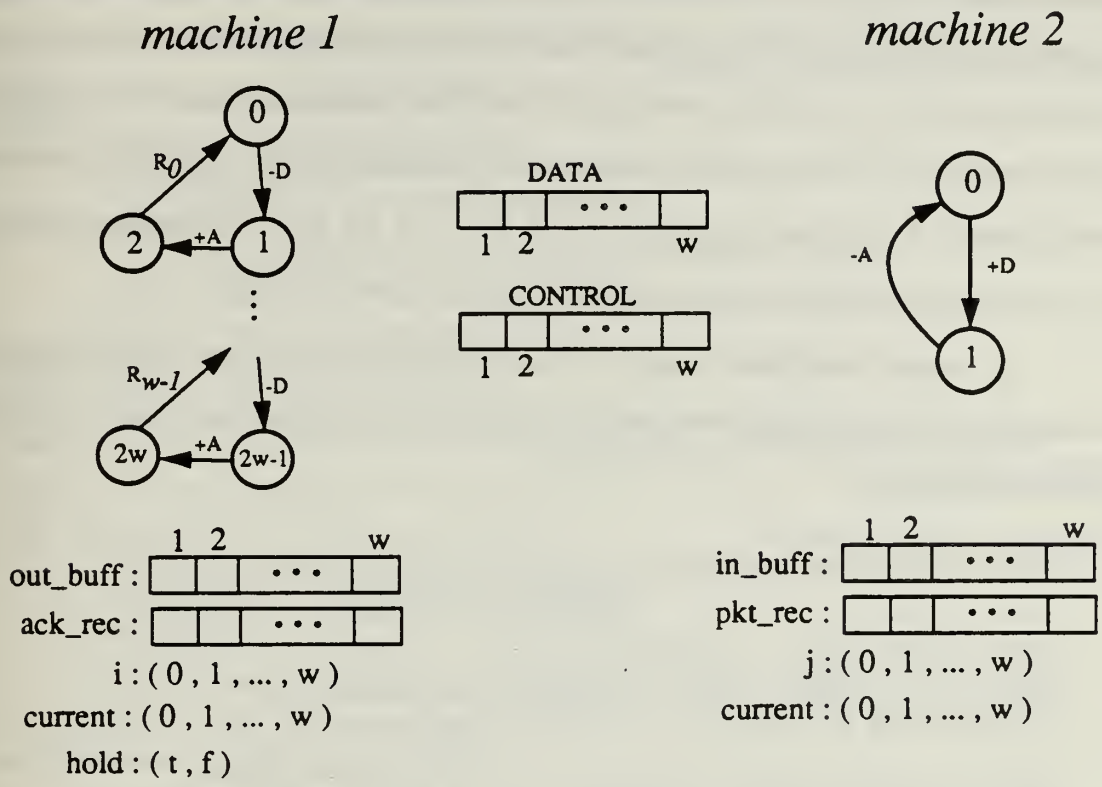
2. Selective Repeat

The next analysis is the selective repeat protocol.The specification defined in [BENV 91]and [STAL 91] has been modified as follows. There are two machines in the system, a sender(m_1) and a receiver (m_2). The senders (m_1) initial state is 0. Two assumptions were made for the analysis. First, all the packets transmitted were received without error and second, no packets were lost or reordered during the transmission.

The specification for the sender is found in Figure 38. As the buffer manager places data in the next available sequence number, the sender places the packet on the channel and increments the index for the next packet to be transmitted. As long as the next packet is not empty, the sender will continue this process until the bottom state on the finite state machine is reached, indicating the transmission of a full window. Acknowledgments (ACK) are passed to the transmitter as they are received. If an ACK is received then the transmitter must determine if the window may be opened and if so, how far. If the ACK is not for the first packet in the window then the flag ack_rec is set, indicating that the packet was received correctly. The window is not advanced because packets that were transmitted earlier are still outstanding. The sequence number within each ACK represents the actual sequence number of the packet received and not the sequence number of the next expected packet, as is common in many protocols.

When an ACKs for the first packet in the window is received the machine clears its buffer, advances the window, and looks at the next sequence number. If the packet has not been received, then that becomes the beginning of the window. If it has been received then the next sequence number is examined until the earliest outstanding packet is found or

the window is fully opened. ACKs that do not correspond to any of the sequence numbers within the current window are ignored.



Transition	Enabling Predicate	Action
-D	*out_buffer(i) /= E ∧* *hold = f*	*DATA(i) :=out_buffer(i)* *inc(i)* *if(i=w) hold := t*
+A	*ack_rec(i) =f ∧* *CONTROL=A(i)*	*ack_rec(i) := t* *CONTROL(i) := E*
R_k $(0 \le k < w)$	*true*	*ack_rec(i) := f* *hold := f*
-A	*CONTROL(j)=E ∧* *pkt_rec(j)=t*	*CONTROL(j) := A(j)* *pkt_rec(j) :=f* *in_buff(j) := e, inc(j)*
+D	*pkt_rec(j)=f*	*in_buff(j) := DATA(j)* *DATA(j) := E* *pkt_rec(j) :=t*

Figure 38: SCM specification, *Selective Repeat*, window size of 1..w.

The receiver as shown in Figure 38 follows the specification given in the predicate action table. The initial state of the receiving machine is 0. Any packets that are received with sequence numbers outside of the window are dropped. If a valid data packet is received then the $+D$ is taken, based upon whether the sequence number of the received packet is equal to i_s. If the sequence number is not i_s then the flag *pkt_rec* is set to *t* and the packet is stored. If it is equal to i_s, then the *pkt_rec* is set, the packet is released to buffer, and i_s is incremented until a sequence number with *pkt_rec=f* is found.

A sample interaction using the program for the analysis of selective repeat, w=1,consists of input files and an output file. The variable definitions contained in the user specification file are:

```
package definitions is
   type scm_transition_type  is (snd_data, rcv_data,
                                 snd_ack,  rcv_ack,
                                 adv_win1, unused);
   type buffer_type   is (d1,e,a1 );
   type boolean_type    is (t,f);
   subtype ack_buffer_type is buffer_type range e..a1;
   subtype data_buffer_type is buffer_type range d1..e;

   type ack_array_type is array(1..1) of ack_buffer_type;
   type data_array_type is array(1..1) of data_buffer_type;

   type boolean_array_type is array(1..1) of boolean_type;

   type machine1_state_type is
      record
         state_number      : natural           := 0;
         out_buffer        : data_array_type   := (d1);
         ack_rec           : boolean_array_type  := (others=>f);
         current           : integer range 1..1 := 1;
         hold              : boolean_type       := f;
   end record;

   type machine2_state_type is
      record
         state_number      : natural                 := 0;
         in_buffer         : data_array_type         := (others=>e);
         pkt_rec           : boolean_array_type      := (others=>f);
   end record;

   type global_variable_type is
      record
         DATA              : data_array_type     := (others=>e);
         CONTROL           : ack_array_type      := (others=>e);
    end record;

end definitions;
```

a. Input of predicate analysis.

The analyze predicate functions contained in the predicate_action.a file are:

```
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type)
return transition_stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if (local.out_buffer(1) /= E) then
        Push(transition_stack,snd_data1);
      end if;
      if ((local.ack_rec(1)=f) and GLOBAL.DATA=A1) then
        Push(transition_stack,rcv_ack1);
      end if;
      Push(transition_stack,adv_win1);
      return transition_stack;
end Analyze_Predicates_Machine1;

function Analyze_Predicates_Machine2(local : machine2_state_type;
                                     GLOBAL: global_variable_type) return
transition_stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA = D1) and (local.pkt_rec(1)=f)) then
        Push(transition_stack,rcv_data1);
      end if;
      if (local.pkt_rec(1)=t) then
        Push(transition_stack,snd_ack1);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2;
```

b. Input of action table.

The action procedure is also a separate compilation unit contained in the predicate_action.a file:

```
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
        when (snd_data1) =>
          out_system_state.GLOBAL_VARIABLES.DATA :=
            in_system_state.machine1_state.out_buffer(1);
        when (rcv_ack1) =>
            out_system_state.machine1_state.ack_rec(1) := t;
            out_system_state.GLOBAL_VARIABLES.DATA := e;
            out_system_state.machine1_state.current := 1;
        when (rcv_data1) =>
            out_system_state.machine2_state.in_buffer(1) :=
              in_system_state.GLOBAL_VARIABLES.DATA;
            out_system_state.GLOBAL_VARIABLES.DATA := e;
```

```
            out_system_state.machine2_state.pkt_rec(1):= t;
        when (snd_ack1) =>
            out_system_state.GLOBAL_VARIABLES.DATA := a1;
            out_system_state.machine2_state.pkt_rec(1) := f;
            out_system_state.machine2_state.in_buffer(1) := e;
        when (adv_win1)   =>
            out_system_state.machine1_state.ack_rec
               (in_system_state.machine1_state.current) := f;
        when others =>
           put_line("There is an error in the Action procedure");
     end case;
end Action;
```

c. Input of finite state machines.

And, finally, the input file for the finite state machines is:

```
start
machine 1
state 0
trans snd_data1 1
state 1
trans rcv_ack1 2
state 2
trans adv_win1 0
machine 2
state 0
trans rcv_data1 1
state 1
trans snd_ack1 0
initial_state 0 0
finish
```

d. Output of analysis.

The output of the analysis is:

```
REACHABILITY ANALYSIS of : sel_rep_w1

Global State GRAPH

0  [ 0 , 0 , D1 , F  , E , F  , E  ]     snd_data1  1
1  [ 1 , 0 , D1 , F  , E , F  , D1 ]     rcv_data1  2
2  [ 1 , 1 , D1 , F  , D1 , T  , E  ]    snd_ack1   3
3  [ 1 , 0 , D1 , F  , E , F  , A1 ]     rcv_ack1   4
4  [ 2 , 0 , D1 , T  , E , F  , E  ]     adv_win1   0

System State GRAPH

0  [ 0 , 0 ]   snd_data1   [ 1 , 0 ] 1
1  [ 1 , 0 ]   rcv_data1   [ 1 , 1 ] 2
2  [ 1 , 1 ]   snd_ack1    [ 1 , 0 ] 3
3  [ 1 , 0 ]   rcv_ack1    [ 2 , 0 ] 4
4  [ 2 , 0 ]   adv_win1    [ 0 , 0 ] 0
```

Machine 1 Array Contents			
From	To	Transition	Executed
0	1	snd_data1	yes
1	2	rcv_ack1	yes
2	0	adv_win1	yes

Machine 2 Array Contents			
From	To	Transition	Executed
0	1	rcv_data1	yes
1	0	snd_ack1	yes

e. System state analysis.

The system state analysis for a window size of 1 is shown in Figure 39.The subscripts are used so that distinct system states having the same tuple may easily be distinguished. The convention is that the subscript is initially 0, and is increased whenever a -*A* transition is taken, by the number of messages which are being acknowledged The analysis for *w=2* is shown in Figure 40. As previously pointed out initial states and variable values are the same as for the w=1, however there are clearly more states in the analysis.

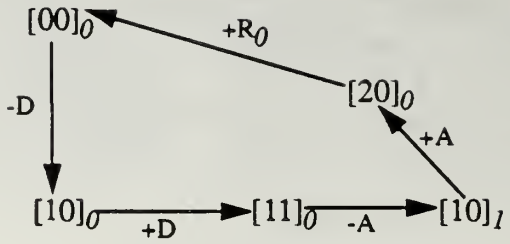


Figure 39: SCM, system state analysis, *Selective Repeat*, w=1.

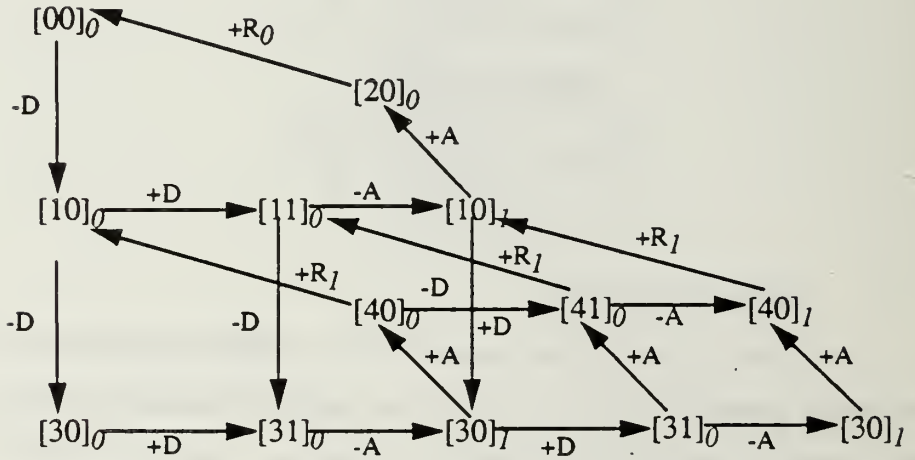


Figure 40: SCM, system state analysis, *Selective Repeat*, w=2.

The graphs contained in Figure 39 and Figure 40 are the basis to the definition of a window size of *w*. The graphs, *SR1(w)* for a nonnegative integer w is a labeled, directed graph, defined by the tuple (N, E, L, Φ), where $N = \{ (x, y, z) \mid (0 \le z \le w, z \le x \le 2w, 0 \le y \le x - z) \}$ is a finite set of nodes, where each node is specified by an ordered triple; $L=\{-D, +D, -A, +A, R_0, \ldots, R_{w-1}\}$ is a finite set of label; the set E of edges is a set of ordered pairs $((x_1, y_1, z_1), (x_1, y_2, z_2))$ of nodes from N, and is the union of the following four sets:

$E_1 = \{((x, y, z), (x+w, y, z)) / (x, y, z) \in N,\ x < w\}$

$E_2 = \{((x, y, z), (x,\ 1, 0)) / (x, y, z) \in N,\ y + z < x\}$

$E_3 = \{((x, y, z), (x,\ y\text{-}1,\ z+1)) / (x, y, z) \in N,\ x > z\}$

$E_4 = \{((x, y, z), (x+1, y, z)) / (x, y, z) \in N,\ y + z < x\}$

$E_5 = \{((x, y, z), (x\text{-}(w+k), y, z)) / (x, y, z) \in N,\ x - (y + z) \leq k\}$

and the mapping $\Phi(L \leftarrow E)$ is defined as follows:

$\forall (x, y, z) \in E_1, \Phi(x, y, z) = -D$

$\forall (x, y, z) \in E_2, \Phi(x, y, z) = +D$

$\forall (x, y, z) \in E_3, \Phi(x, y, z) = -A$

$\forall (x, y, z) \in E_4, \Phi(x, y, z) = +A$

$\forall (x, y, z) \in E_5, \Phi(x, y, z) = R_k, k = |w - x|$

As with the go_back_n analysis, each node of the graph can be thought of as a point in 3-dimensional space, with nonnegative, integral coordinates (*x,y,z*). The structure of the graph is a sequence of *w+1* triangles, one on top of the other, with the largest triangle at the bottom and the smallest is a single point at the top level.

Let f(w) be the amount of nodes in a system state graph. The lemmas that support it, for the selective repeat protocol are found in [JENS 92]. The graph SR1(w) has $f(w) = w^2 + (w+1)^2$. The size of the graphs according to window size are:

w	f(w)
0	1
1	5
2	13
3	25
4	41

Test runs were done for and the amount of nodes in each graph were consistent with the table above. The specification input files and the output is in Appendices I and J.

VI. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a program was introduced that analyzed network protocols using the CFSM and SCM models. The program was successfully developed in an Ada environment. The Ada tools that made implementing the models easier were encapsulation, information hiding, generic programming units.

In Ada, network protocol specifications can be represented in an intuitive manner. The finite state machines and the associated predicate action tables were converted to Ada language parameters for the analysis. The language environment enforces the rules of the protocol as well as the allowable behavior of all the variables. Dynamic construction of reachability graphs allowed the user to determine how large or small an analysis should be. The protocol designer or engineer that uses this program can quickly become familiar with the behavior of the protocol by simply constructing the Ada specification. It is interesting to note that due to the automated specification analysis some previous work using the SCM model has since been modified. The analysis provides information on occurrences of deadlock, unspecified reception, unexecuted transition, and message flow exceeding channel capacity.

The programming environment provided an adequate platform to develop the program. The ability to use encapsulation, information hiding, tasking, and generic code allowed the program to be developed in a step-wise, compartmented fashion. The availability of a powerful debugger (DBX) enhanced the transition from developing to testing the program. To allow for a more transportable product this program might be converted to C or C++, so that others may benefit from its use.

The program was validated with previous work done on widely used protocols. In Lundy's papers [LUND 88], [LUND 91a], [LUND 91b], [LUND 92a] and [LUND 92b] a number of protocols were manually analyzed using the SCM model. A subset of those analysis were performed using the program, achieving identical results.It is interesting to note that due to the automated analysis of previous work using the SCM model, some

specifications have had to be modified. In must also be noted that example analysis were only done on two machine specifications.

There are several questions and areas open for further work which remain. An important step would be to expand the program to allow for more than two machines. Although most protocols can be modelled with two machines, it is a realistic requirement to model three or more. The program could be made more interactive with the user, allowing the user to change the specification real time when an error occurs. The program was developed for use on a workstation, future work could concentrate on PC versions of Ada or C/C++. A picture is worth a thousand words; what is done textually can sometimes be represented better graphically. A graphical user interface would enhance the users ability to specify a protocol and understand the analysis.

It is important to say that developing this program was FUN. One of the features of this automated tool is its understandability. By developing an automated tool it became apparent that the user needs to feel comfortable with the language used as well as how a specification looks and feels when input for analysis. The results provided by the many runs of this program focused attention to where it needed to be- the protocol behavior, not the programming language anomalies.

APPENDIX A CFSM CODE

The program listing begins on the following page.

```
-------------------------------------------------------------------------------------------------------
-- Title       : main
-- File        ; main.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This is the main procedure.  The user provides a text file that is the specification of a two
--               machine model.  The text file is parsed and machine adjacency lists are built.  A reachability
--               analysis is done on the machine specification and output to screen and a text file. All
--               procedures that the main uses were created and compiled as separate units. Desriptions of each
--               function and procedure can be found in the header of  the associated units/files.
-------------------------------------------------------------------------------------------------------


with Text_IO, queues, stacks;
use  Text_IO;

procedure main is

   max_length : natural := 40;                    --length of global state array
   type cfsm_instruction_type is (start, number_of_machines, machine, number_of_states, state, initial_state, trans,
    finish);
   type cfsm_transition_type  is (snd, rcv, unused);
   type visit_type            is (yes, no);
   type search_type           is (dfs, bfs);
   package trans_enum_io is new Text_IO.Enumeration_IO(cfsm_transition_type);
   package instr_enum_io is new Text_IO.Enumeration_IO(cfsm_instruction_type);
   package vst_enum_io is new Text_IO.Enumeration_IO(visit_type);
   package srch_enum_io is new Text_IO.Enumeration_IO(search_type);
   package cfsm_integer_io is new Integer_IO(integer);
   use  trans_enum_io, instr_enum_io, vst_enum_io, cfsm_integer_io;
   ------------------------------------------------------------------------
   -- data structures for the machine arrays
   ------------------------------------------------------------------------
   package queue_pack is new queues(character, MAX => 3);
   use  queue_pack;
   type machine_array_record_type;
   type Slink_type is access machine_array_record_type;
   type machine_array_record_type is
      record
         transition     : cfsm_transition_type := unused;
         message        : character  := ' ';
         next_Mstate    : natural    := 0;
         visited        : visit_type := no;
         Slink          : Slink_type := null;
   end record;

   type machine_array_type is array(positive range 1 .. 10) of Slink_type;

   type system_array_type is array(1 .. 2) of machine_array_type;

   ------------------------------------------------------------------------
   --data structures for the global state tuple(node)
```

```
--------------------------------------------------------------------------
type global_state_type;
type Glink_type is access global_state_type;

--------------------------------------------------------------------------
--transition structure
--------------------------------------------------------------------------
type Gstate_transition_type is
   record
      Gtransition : cfsm_transition_type := unused;
      Gmessage    : character            := ' ';
      new_node    : boolean              := true;
      Glink       : Glink_type;
end record;

--------------------------------------------------------------------------
--global tuple structure
--------------------------------------------------------------------------
type Gstate_record_type is
   record
      machine1_state : natural := 0;
      machine2_state : natural := 0;
      queue_12       : queue_pack.Queue;  --array of characters
      queue_21       : queue_pack.Queue;
end record;

--------------------------------------------------------------------------
--Global state node, contains transition, state, and link information
--needed for building the global state graph
--------------------------------------------------------------------------
type global_state_type is
   record
      system_state_number : natural := 0;
      Gtuple : Gstate_record_type;
      link1  : Gstate_transition_type;
      link2  : Gstate_transition_type;
      link3  : Gstate_transition_type;
      link4  : Gstate_transition_type;
end record;

--------------------------------------------------------------------------
--declaration of stacks of pointers and queues of pointers
--along with queues of messages to simulate two way communications
--between machines
--------------------------------------------------------------------------
package Gpointer_stack_pack is new stacks(Glink_type, MAX => 10);
package Gstate_stack_pack is new stacks(global_state_type);
package Gpointer_queue_pack is new queues(Glink_type, MAX => 10);
use  Gpointer_stack_pack, Gstate_stack_pack, Gpointer_queue_pack;
```

```
------------------------------------------------------------------------
--global variables are initialized
------------------------------------------------------------------------
initial_global_state : Glink_type;              -- top tuple in global graph
cfsm_instruction     : cfsm_instruction_type;   --used in case statements
total_machines       : natural := 0;            --number of machines in the system
total_states         : natural := 0;            --number of states per machine
current_state        : natural := 0;            --current_state of the current machine
next_state           : natural := 0;
current_machine      : natural := 0;
current_transition   : cfsm_transition_type;
search               : search_type;             --values dfs(depth first) and bfs(breadth first)
current_message      : character;               --message off the top of the queue
current_queue12      : queue_pack.Queue;        --array of characters(string)
current_queue21      : queue_pack.Queue;
start_state_1        : natural := 0;
start_state_2        : natural := 0;
start_state_3        : natural := 0;
temp_Gstate          : Gstate_record_type;      -- a temporary holding place for tuples
machine_array        : system_array_type;       -- an array of arrays so that the machines can be subscripted to be
                                                -- identified
Gpointer_stack       : Gpointer_stack_pack.stack;
Gpointer_queue       : Gpointer_queue_pack.queue;
number_of_nodes      : natural := 0;
output_line_count    : natural := 1;            --stops output from scrolling if longer than 25 lines
file_name            : string(1 .. 20) := (others => ' ');
out_file_name        : string(1 .. 20) := (others => ' ');
analysis             : string(1 .. 20) := (others => ' '); --string that holds the type of analysis(dfs,bfs)
in_name_len          : integer;                  --variables that holds length of a string
out_name_len         : integer;
analysis_len         : integer;
cfsm                 : file_type;                --input file pointer
reach                : file_type;                --output file pointer

----------------------------output.a-----------------------------------------
procedure pause_output                  is separate;
procedure unprompted_pause              is separate;
procedure PrintQueue                    (Q: in queue_pack.Queue) is separate;
procedure output_Gstate_node            (Gstate_pointer : in out Glink_type; Error_flag : in out boolean) is separate;
procedure output_Gstate_transition      (Gstate_pointer : in out Glink_type) is separate;
procedure output_Gstates                (top_Gstate : in out Glink_type; number_of_nodes : in out natural) is separate;
procedure output_machine_arrays         (machine_array : in out system_array_type) is separate;
procedure PrintQueue_to_file            (Q : in queue_pack.Queue) is separate;
procedure output_Gstate_node_to_file    (Gstate_pointer : in out Glink_type; Error_flag : in out boolean) is separate;
procedure output_Gstate_transition_to_file(Gstate_pointer : in out Glink_type) is separate;
procedure output_Gstates_to_file        (top_Gstate : in out Glink_type; number_of_nodes : in out natural) is separate;
procedure output_machine_arrays_to_file (machine_array : in out system_array_type) is separate;
procedure create_output_file            is separate;

----------------------------input.a------------------------------------------
procedure load_machine_array(number     : in out natural;
                             state      : in out natural;
                             Next_state : in out natural;
```

```
                                Transition : in out cfsm_transition_type;
                                Message    : in out character) is separate;
   procedure read_in_file(cfsm : in out file_type) is separate;

   ----------------------------reachability.a-------------------------------------
   function IsEqual(tuple1 : in Gstate_record_type; tuple2 : in Gstate_record_type) return boolean is separate;

   ----------------------------search.a-------------------------------------------
   function search_for_tuple(top : in Glink_type; tuple : in Gstate_record_type) return Glink_type is separate;

   ----------------------------reachability.a-------------------------------------
   procedure build_Gstate_graph(machine_array   : in out system_array_type;
                                start_state_1   : in out natural;
                                start_state_2   : in out natural;
                                top_Gstate      : in out Glink_type;
                                number_of_nodes : in out natural) is separate;
   procedure output_analysis    is separate;
   procedure clear_pointers     is separate;

   ------------------------------------------------------------------------------

begin
   main_cfsm_loop :
      loop
         put(ASCII.ESC);     --call to operating system to clear the screen
         put_line("[2J");
         new_line(10);

         --query the user for the input file
         loop
            begin
               set_col(10);
               put("Filename of cfsm specification('Q' to quit): ");
               get_line(file_name, in_name_len);
               if (file_name(1) = 'Q') then
                  exit main_cfsm_loop;
               else
                  open(cfsm, mode => in_file, name => file_name(1 .. in_name_len));
               end if;
               exit;
            exception
               when NAME_ERROR =>
                  put_line("That file does not exist. Reenter!!");
            end;
         end loop;
         ---------------------------------------------------------------
         --query the user for the way that the global analysis graph is to
         --be built.  A stack based construction will lead to a breadth first
```

```
        --construction and a queue based construction will lead to a depth
        --first construction.
        ------------------------------------------------------------------
        loop
           begin
              set_col(10);
              put("Breadth First (B) or Depth First (D) Analysis: ");
              get_line(analysis, analysis_len);
              if ((analysis(1) = 'B') or (analysis(1) = 'b')) then
                 search := bfs;
              elsif ((analysis(1) = 'D') or (analysis(1) = 'd')) then
                 search := dfs;
              else
                 raise NAME_ERROR;
              end if;
              exit;
           exception
              when NAME_ERROR =>
                 put_line("Not a correct choice. Enter a 'B' or a 'D'!!");
           end;
        end loop;
        create_output_file;
        read_in_file(cfsm);
        output_analysis;
        close(cfsm);
        close(reach);
        clear_pointers;
     end loop main_cfsm_loop;
end main;
```

```
separate(main)
--=============================================================================================================
-- Title       : load_machine_array
-- File        : input.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Builds machine adjacency list.  Takes transition and message as input and adds them to
--               the lists.  The possible transitions are 'send' or 'receive'.  The messages are
--               simulated by a character identifier.  The machine arrays are represented by a single
--               one dimensional array of arrays of linked lists.
--=============================================================================================================

procedure load_machine_array(number     : in out natural;
                             state      : in out natural;
                             Next_state : in out natural;
                             Transition : in out cfsm_transition_type;
                             Message    : in out character) is

   end_of_list : Slink_type := machine_array(number) (state);
   temp_link   : Slink_type := new machine_array_record_type;
begin

   temp_link.transition := Transition;
   temp_link.message := Message;
   temp_link.next_Mstate := Next_state;
   if (end_of_list = null) then
      machine_array(number) (state) := temp_link;
   else
      while (end_of_list.Slink /= null) loop
         end_of_list := end_of_list.Slink;
      end loop;
      end_of_list.Slink := temp_link;
   end if;
end load_machine_array;
```

```
separate(main)
-------------------------------------------------------------------------------------------------
-- Title       : read_in_file
-- File        : input.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Parses input file one line at a time. Each line is read into a line buffer and tokens
--               are extracted from the line.  A token may be an understandable machine instruction:
--
--                 start:             placed as a flag at the beginning of the file
--                 number_of_states: a variable used to tell the dimensions of
--                                   machine array.
--                 number_of_machines: same as above, this variable is not used
--                                     but built in to use for more dynamic versions
--                 machine:          a flag to identify the array that the machine
--                                   instructions are to be loaded.
--                 state:            identifies the row of the machine array that
--                                   the instruction and message are to be loaded
--                 trans:            identify a transition <trans -X 2> means
--                                   transition from the current machine and
--                                   state that is a send with message X to
--                                   state 2.
--                 intial_state:     identifies the starting state of machines.
--                                   <initial_state 1 1> means that the starting
--                                   state of machine one is state one and the
--                                   starting state of machine two is state one.
--                 finish:           a end of text file delimeter.
--
--               As the file is scanned and the text is converted to usable tokens the tokens
--               are output to default output(CRT) for the user to view. The tokens determine
--               how the machine adjacency lists are built.
--
-------------------------------------------------------------------------------------------------

procedure read_in_file(cfsm : in out file_type) is

   line_buffer : string(1 .. 80) := (others => ' ');
   token       : string(1 .. 80) := (others => ' ');
   token_index : integer         := 1;
   line_length : integer;
   line_index  : integer         := 1; --variable used for halting output every
   enum_length : integer;              --25 lines.
   conv_length : integer;
   line_count  : integer         := 1;

   --clears the token used in parsing
   procedure clear_token(in_token : in out string) is
   begin
      for i in 1 .. 80 loop
         in_token(i) := ' ';
      end loop;
```

```
   end clear_token;

begin
   new_line(4);

   while not End_of_File(cfsm) loop
      line_count := line_count + 1;
      if ((line_count mod 20) = 0) then
         pause_output;
      end if;
      get_line(cfsm, line_buffer, line_length);

      ----begin parsing line get rid of leading blanks
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;

      ---read in instruction token and cast string to enumerated type
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;

      --cast a string into an enumerated type
      instr_enum_io.get(token(1 .. token_index - 1), cfsm_instruction, enum_length);
      token_index := 1;
      clear_token(token);
      case cfsm_instruction is
         when start =>
            put_line("            PROGRAM IS LOADING.....");
            new_line(2);
            set_col(20);
            put(cfsm_instruction, set => lower_case);
            line_index := 1;

         when number_of_machines =>
            set_col(20);
            put(cfsm_instruction, set => lower_case);
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;

            get(token, total_machines, conv_length);
```

```
      put(total_machines, width => 2);
      line_index := 1;
      token_index := 1;
      clear_token(token);

   when number_of_states =>
      set_col(20);
      put(cfsm_instruction, set => lower_case);
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;

      get(token, total_states, conv_length);
      put(total_states, width => 2);
      line_index := 1;
      token_index := 1;
      clear_token(token);

   when machine =>
      set_col(20);
      put(cfsm_instruction, set => lower_case);
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;

      get(token, current_machine, conv_length);
      put(current_machine, width => 2);
      line_index := 1;
      token_index := 1;
      clear_token(token);

   when state =>
      set_col(20);
      put(cfsm_instruction, set => lower_case);
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;

      get(token, current_state, conv_length);
```

```
      put(current_state, width => 2);
      line_index := 1;
      token_index := 1;
      clear_token(token);

   when trans =>
      set_col(20);
      put(cfsm_instruction, set => lower_case);
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;
      if token(1) = '-' then
         current_transition := snd;
      else
         current_transition := rcv;
      end if;
      current_message := token(2);
      clear_token(token);
      token_index := 1;
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;
      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;
      get(token, next_state, conv_length);
      clear_token(token);
      token_index := 1;
      line_index := 1;
      if current_transition = snd then
         put(" -");
      else
         put(" +");
      end if;
      put(current_message);
      put(next_state, width => 2);
      load_machine_array(current_machine, current_state, next_state, current_transition, current_message);
   when initial_state =>
      set_col(20);
      put(cfsm_instruction, set => lower_case);
      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
```

```
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;
            get(token, start_state_1, conv_length);
            put(start_state_1, width => 2);
            token_index := 1;
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;
            get(token, start_state_2, conv_length);
            put(start_state_2, width => 2);
            line_index := 1;
            token_index := 1;
            clear_token(token);

         when finish =>
            set_col(20);
            put(cfsm_instruction, set => lower_case);
            new_line(2);
            put_line("       cfsm's are  loaded!!!");
            exit;

         when others =>
            put_line("                 error---error---error");
            for x in 1 .. line_length loop
               put(line_buffer(x));
            end loop;

      end case;
      new_line;
   end loop;
end read_in_file;
```

```
--=============================================================================================================
-- Title       : queues
-- File        : queues.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This generic package contains procedures and functions to implement a queue of
--               type item.  In the model there are queues of characters (to simulate communication
--               channels) and queues of global tuple pointers to aid in the construction
--               of the global reachability graph.  The variables chosen for the procedures and functions
--               are self descriptive.  The default size of a queue is 10, the user has the option of
--               making the queue bigger ( up to the maximum size of the supporting hardware.) A number
--               of the functions were taken from the text, "Data Structures in Ada" by Michael Feldman.
--=============================================================================================================


generic

   type Item is private;
   MAX : in integer := 10;

package queues is

   type ArrayList is array(1 .. MAX) of Item;
   type Queue is
      record
         Store : ArrayList;
         tail  : integer range 0 .. MAX := 0;
   end record;

   procedure MakeEmpty(Q : in out Queue);
   procedure Enqueue(Q : in out Queue; E : in Item);
   procedure Dequeue(Q : in out Queue);
   function First(Q : Queue) return Item;
   function IsEmpty(Q : Queue) return boolean;
   function IsEqual(Q1 : Queue; Q2 : Queue) return boolean;

   QueueFull  : exception;
   QueueEmpty : exception;

end queues;

package body queues is

   procedure MakeEmpty(Q : in out queue) is
      blank_line : string(1 .. 80) := (others => ' ');
   begin
```

```
      Q.tail := 0;
   end MakeEmpty;

   procedure Enqueue(Q : in out Queue; E : in Item) is
   begin
      if Q.tail = MAX then
         raise QueueFull;
      else
         Q.tail := Q.tail + 1;
         Q.Store(Q.tail) := E;
      end if;
   end Enqueue;

   procedure Dequeue(Q : in out Queue) is
   begin
      if Q.tail = 0 then
         raise QueueEmpty;
      else
         for i in 2 .. Q.tail loop
            Q.Store(i - 1) := Q.Store(i);
         end loop;
         Q.tail := Q.tail - 1;
      end if;
   end Dequeue;

   function First(Q : Queue) return Item is
   begin
      if Q.tail = 0 then
         raise QueueEmpty;
      else
         return Q.Store(1);
      end if;
   end First;

   function IsEmpty(Q : Queue) return boolean is
   begin
      return Q.tail = 0;
   end IsEmpty;

   function IsEqual(Q1 : Queue; Q2 : Queue) return boolean is
   begin
      for i in 1 .. Q1.tail loop
         if (Q1.Store(i) /= Q2.Store(i)) then
            return false;
         end if;
      end loop;
      return true;
   end IsEqual;

end queues;
```

```
--=============================================================================================
-- Title       : stacks
-- File        : stacks.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This generic package contains procedures and functions to implement a stack of
--               type item.  In the model there are only stacks of pointers to aid in the construction
--               of the global reachability graph.  The variables chosen for the procedures and functions
--               are self descriptive.  The default size of a queue is 10, the user has the option of
--               making the queue bigger ( up to the maximum size of the supporting hardware.) A number
--               of the functions were taken from the text, "Data Structures in Ada" by Michael Feldman.
--=============================================================================================

generic

   type Item is private;
   MAX : in integer := 10;

package stacks is

   type ArrayList is array(1 .. MAX) of Item;
   type Stack is
      record
         Store  : ArrayList;
         latest : integer range 0 .. MAX := 0;
   end record;

   procedure MakeEmpty(S : in out Stack);
   function IsEmpty(S : Stack) return boolean;
   procedure Push(S : in out Stack; E : in Item);
   procedure Pop(S : in out Stack; E : out Item);
   function Top(S : Stack) return Item;

   StackFull  : exception;
   StackEmpty : exception;

end stacks;

package body stacks is

   procedure MakeEmpty(S : in out Stack) is
   begin
      S.latest := 0;
   end MakeEmpty;
```

```
    procedure Push(S : in out Stack; E : in Item) is
    begin
       if S.latest = MAX then
          raise StackFull;
       else
          S.latest := S.latest + 1;
          S.Store(S.latest) := E;
       end if;
    end Push;

    procedure Pop(S : in out Stack; E : out Item) is
    begin
       if S.latest = 0 then
          raise StackEmpty;
       else
          E := S.Store(S.latest);
          S.latest := S.latest - 1;
       end if;
    end Pop;

    function Top(S : Stack) return Item is
    begin
       if S.latest = 0 then
          raise StackEmpty;
       else
          return S.Store(S.latest);
       end if;
    end Top;

    function IsEmpty(S : Stack) return boolean is
    begin
       return S.latest = 0;
    end IsEmpty;

end stacks;
```

```
separate(main)
-------------------------------------------------------------------------------------------------
-- Title       : IsEqual
-- File        : reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Compares two system state tuples for equality.  Returns a boolean(true/false).
-------------------------------------------------------------------------------------------------

function IsEqual(tuple1 : in Gstate_record_type; tuple2 : in Gstate_record_type) return boolean is

begin
   if ((tuple1.machine1_state = tuple2.machine1_state) and then (tuple1.machine2_state = tuple2.machine2_state) and then
    (tuple1.queue_12.tail = tuple2.queue_12.tail) and then (tuple1.queue_21.tail = tuple2.queue_21.tail)) then
      if (tuple1.queue_12.tail = 0) then
         if (tuple1.queue_21.tail = 0) then
            return true;
         else
            if (IsEqual(tuple1.queue_21, tuple2.queue_21)) then
               return true;
            else
               return false;
            end if;
         end if;
      else
         if (tuple1.queue_21.tail = 0) then
            if (IsEqual(tuple1.queue_12, tuple2.queue_12)) then
               return true;
            else
               return false;
            end if;
         else
            if ((IsEqual(tuple1.queue_12, tuple2.queue_12)) and then (IsEqual(tuple1.queue_21, tuple2.queue_21))) then
               return true;
            else
               return false;
            end if;
         end if;
      end if;
   else
      return false;
   end if;
end IsEqual;
```

```
separate(main)
--=============================================================================================
-- Title       : build_Gstate_graph
-- File        : reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure builds the system reachability graph for two machines.  Each system
--               state node can have up to 4 transitions.  The algorithm for this procedure is
--               as follows.  Each machine state is checked for valid transitions to another state.
--               All send transitions are checked and all receive transitions are checked.  A temporary
--               tuple is constructed to hold the values of each machines current state and the queue
--               contents.  A BFS/DFS search is done of the graph, checking all nodes for the same values
--               in temp tuple to see if it already exists. If it does, a link is made from the current
--               node to the found node. If a match for the temp is not found a node is created and
--               inserted into the graph. A pointer to the newly added node is added to the pointer
--               queue/stack.  A pointer is retrieved from the stack/queue after a scan is done from each
--               machine array. This procedure terminates whenever there are no transitions out of any
--               machine array states, or the intial state for each machine is reached, or the pointer
--               queue/stack is empty.  As can be demonstrated, CFSM reachability graphs can grow without
--               bounds. A max pointer queue/stack size is introduced to eliminate unboundedness. Variables
--               accepted by this procedure are machine states, machine queue contents, number of nodes,
--               currently in the graph, and the pointer to the head of the graph.
--=============================================================================================

procedure build_Gstate_graph(machine_array   : in out system_array_type;
                             start_state_1   : in out natural;
                             start_state_2   : in out natural;
                             top_Gstate      : in out Glink_type;
                             number_of_nodes : in out natural) is
   parent_Gstate, current_Gstate, find_Gstate, temp_tuple_pointer : Glink_type := null;
   Gstate_counter                                                 : natural    := 1;
   machine_1                                                      : natural    := 1;
   machine_2                                                      : natural    := 2;
   M1_state, M2_state                                             : natural;    --machine 1 and machine 2 state holder
   next_state                                                     : natural;    --returned value from a transition
                                                                                --search
   place_holder1                                                  : Slink_type;
   place_holder2                                                  : Slink_type;
   initial_Gstate                                                 : Gstate_record_type;
   is_new                                                         : boolean;
```

```
--=============================================================================================
-- Title       : Link_Gstate
-- File        : reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure finds the next non-null pointer to link a new global state node too.
--               If all transitions are used, a message is output to the screen.  In this case another
--               model may be more appropriate, or the current procedure can be modified to account
--               for more links.
--=============================================================================================
procedure Link_Gstate(temp           : in out Gstate_record_type;
                      trans          : in out cfsm_transition_type;
                      mess           : in out character;
                      current_Gstate : in out Glink_type;
                      next_Gstate    : in out Glink_type;
                      new_one        : in out boolean) is
begin
   if (current_Gstate.link1.Glink = null) then
      current_Gstate.link1.Gmessage := mess;
      current_Gstate.link1.Gtransition := trans;
      current_Gstate.link1.Glink := next_Gstate;
      if (not new_one) then
         current_Gstate.link1.new_node := false;
      end if;
   else
      if (current_Gstate.link2.Glink = null) then
         current_Gstate.link2.Gmessage := mess;
         current_Gstate.link2.Gtransition := trans;
         current_Gstate.link2.Glink := next_Gstate;
         if (not new_one) then
            current_Gstate.link2.new_node := false;
         end if;
      else
         if (current_Gstate.link3.Glink = null) then
            current_Gstate.link3.Gmessage := mess;
            current_Gstate.link3.Gtransition := trans;
            current_Gstate.link3.Glink := next_Gstate;
            if (not new_one) then
               current_Gstate.link3.new_node := false;
            end if;
         else
            if (current_Gstate.link4.Glink = null) then
               current_Gstate.link4.Gmessage := mess;
               current_Gstate.link4.Gtransition := trans;
```

```
                current_Gstate.link4.Glink := next_Gstate;
                if (not new_one) then
                   current_Gstate.link4.new_node := false;
                end if;
             else
                put_line("All transition pointers used!!!");
             end if;
          end if;
       end if;
    end if;

 end Link_Gstate;

 ------------------------------------------------------------------------
 --This procedure simply links the newly created global state node to the previous
 --node in the graph.
 ------------------------------------------------------------------------
 procedure Insert_Gstate(temp          : in out Gstate_record_type;
                         trans         : in out cfsm_transition_type;
                         mess          : in out character;
                         current_Gstate : in out Glink_type;
                         counter       : in out natural) is
    temp_node  : Glink_type := new global_state_type;
    temp_ptr   : Glink_type;
    is_new     : boolean;
    Error_flag : boolean;
 begin
    temp_node.system_state_number := counter;
    temp_node.Gtuple := temp;
    is_new := true;
    Link_Gstate(temp, trans, mess, current_Gstate, temp_node, is_new);
    current_Gstate := temp_node;
 end Insert_Gstate;

 ------------------------------------------------------------------------
begin
 temp_tuple_pointer                       := new global_state_type;
 temp_tuple_pointer.system_state_number   := Gstate_counter;    --initialize the tuple counter
 temp_tuple_pointer.Gtuple.machine1_state := start_state_1;     --build initial system tuple
 temp_tuple_pointer.Gtuple.machine2_state := start_state_2;
 top_Gstate                               := temp_tuple_pointer;
 parent_Gstate                            := top_Gstate;        --set the current state pointer to the top
 current_Gstate                           := parent_Gstate;
 M1_state                                 := start_state_1;
 M2_state                                 := start_state_2;
 initial_Gstate                           := temp_tuple_pointer.Gtuple;
 new_line;
 ------------------------------------------------
 main_loop :
    loop
       place_holder1 := machine_array(1) (M1_state);
       while (place_holder1 /= null) loop
          --if the current state in the machine does not have a null value check for valid transitions.
          loop
```

```
begin
   current_Gstate := parent_Gstate;
   current_queue12 := current_Gstate.Gtuple.queue_12;
   current_queue21 := current_Gstate.Gtuple.queue_21;
   if (place_holder1.transition = snd) then
      place_holder1.visited := yes;
      Enqueue(current_queue12, place_holder1.message);
      temp_Gstate.machine1_state := place_holder1.next_Mstate;
      temp_Gstate.machine2_state := M2_state;
      temp_Gstate.queue_12 := current_queue12;
      temp_Gstate.queue_21 := current_queue21;
      find_Gstate := search_for_tuple(top_Gstate, temp_Gstate);
      if (find_Gstate = null) then
         Gstate_counter := Gstate_counter + 1;
         Insert_Gstate(temp_Gstate, place_holder1.transition, place_holder1.message, current_Gstate,
          Gstate_counter);
         case search is
            when bfs =>
               Enqueue(Gpointer_queue, current_Gstate);
            when dfs =>
               Push(Gpointer_stack, current_Gstate);
            when others =>
               put_line("error");
         end case;
      else
         is_new := false;
         Link_Gstate(temp_Gstate, place_holder1.transition, place_holder1.message, current_Gstate,
          find_Gstate, is_new);
      end if;
   else
      if ((not IsEmpty(current_queue21)) and then (place_holder1.transition = rcv and then
       (place_holder1.message = First(current_queue21)))) then
         place_holder1.visited := yes;
         Dequeue(current_queue21);
         temp_Gstate.machine1_state := place_holder1.next_Mstate;
         temp_Gstate.machine2_state := M2_state;
         temp_Gstate.queue_12 := current_queue12;
         temp_Gstate.queue_21 := current_queue21;
         find_Gstate := search_for_tuple(top_Gstate, temp_Gstate);
         if (find_Gstate = null) then
            Gstate_counter := Gstate_counter + 1;
            Insert_Gstate(temp_Gstate, place_holder1.transition, place_holder1.message, current_Gstate,
             Gstate_counter);
            case search is
               when bfs =>
                  Enqueue(Gpointer_queue, current_Gstate);
               when dfs =>
                  Push(Gpointer_stack, current_Gstate);
               when others =>
```

```
                        put_line("error");
                     end case;
                  else
                     is_new := false;
                     Link_Gstate(temp_Gstate, place_holder1.transition, place_holder1.message, current_Gstate,
                      find_Gstate, is_new);
                  end if;
               end if;
            end if;
            place_holder1 := place_holder1.Slink;
            exit;
         exception
            when Gpointer_queue_pack.QueueFull =>
               put("     The pointer queue is full at state: ");
               put(current_Gstate.system_state_number, width => 1);
               new_line;
               put(reach, "     The pointer queue is full at state: ");
               put(reach, current_Gstate.system_state_number, width => 1);
               new_line(reach);
               Gpointer_queue_pack.MakeEmpty(Gpointer_queue);
               exit main_loop;
            when Gpointer_stack_pack.StackFull =>
               put("     The pointer stack is full at state: ");
               put(current_Gstate.system_state_number, width => 1);
               new_line;
               put(reach, "     The pointer stack is full at state: ");
               put(reach, current_Gstate.system_state_number, width => 1);
               new_line(reach);
               Gpointer_stack_pack.MakeEmpty(Gpointer_stack);
               exit main_loop;
            when queue_pack.QueueFull =>
               put("        Exceeded the channel capacity in queue_12 at state: ");
               put(parent_Gstate.system_state_number, width => 1);
               new_line;
               put(reach, "        Exceeded the channel capacity in queue_12 at state: ");
               put(reach, parent_Gstate.system_state_number, width => 1);
               new_line(reach);
               place_holder1 := place_holder1.Slink;
               exit;
         end;
      end loop;
   end loop;
   place_holder2 := machine_array(2) (M2_state);
   while (place_holder2 /= null) loop
      loop
         begin
            current_Gstate := parent_Gstate;
            current_queue12 := current_Gstate.Gtuple.queue_12;
            current_queue21 := current_Gstate.Gtuple.queue_21;
            if (place_holder2.transition = snd) then
               place_holder2.visited := yes;
               Enqueue(current_queue21, place_holder2.message);
               temp_Gstate.machine1_state := M1_state;
               temp_Gstate.machine2_state := place_holder2.next_Mstate;
```

```
            temp_Gstate.queue_12 := current_queue12;
            temp_Gstate.queue_21 := current_queue21;
            find_Gstate := search_for_tuple(top_Gstate, temp_Gstate);
            if (find_Gstate = null) then
               Gstate_counter := Gstate_counter + 1;
               Insert_Gstate(temp_Gstate, place_holder2.transition, place_holder2.message, current_Gstate,
                Gstate_counter);
               case search is
                  when bfs =>
                     Enqueue(Gpointer_queue, current_Gstate);
                  when dfs =>
                     Push(Gpointer_stack, current_Gstate);
                  when others =>
                     put_line("error");
               end case;
            else
               is_new := false;
               Link_Gstate(temp_Gstate, place_holder2.transition, place_holder2.message, current_Gstate,
                find_Gstate, is_new);
            end if;
         else
            if ((not IsEmpty(current_queue12)) and then (place_holder2.transition = rcv and then
             (place_holder2.message = First(current_queue12)))) then
               place_holder2.visited := yes;
               Dequeue(current_queue12);
               temp_Gstate.machine1_state := M1_state;
               temp_Gstate.machine2_state := place_holder2.next_Mstate;
               temp_Gstate.queue_12 := current_queue12;
               temp_Gstate.queue_21 := current_queue21;
               find_Gstate := search_for_tuple(top_Gstate, temp_Gstate);
               if (find_Gstate = null) then
                  Gstate_counter := Gstate_counter + 1;
                  Insert_Gstate(temp_Gstate, place_holder2.transition, place_holder2.message, current_Gstate,
                   Gstate_counter);
                  case search is
                     when bfs =>
                        Enqueue(Gpointer_queue, current_Gstate);
                     when dfs =>
                        Push(Gpointer_stack, current_Gstate);
                     when others =>
                        put_line("error");
                  end case;
               else
                  is_new := false;
                  Link_Gstate(temp_Gstate, place_holder2.transition, place_holder2.message, current_Gstate,
                   find_Gstate, is_new);
               end if;
            end if;
         end if;
```

```
                place_holder2 := place_holder2.Slink;
                exit;
             exception
                when Gpointer_queue_pack.QueueFull =>
                   put("    The pointer queue is full at state: ");
                   put(current_Gstate.system_state_number, width => 1);
                   new_line;
                   put(reach, "    The pointer queue is full at state: ");
                   put(reach, current_Gstate.system_state_number, width => 1);
                   new_line(reach);
                   case search is
                      when bfs =>
                         Gpointer_queue_pack.MakeEmpty(Gpointer_queue);
                      when dfs =>
                         Gpointer_stack_pack.MakeEmpty(Gpointer_stack);
                      when others =>
                         put_line("error");
                   end case;
                   exit main_loop;
                when queue_pack.QueueFull =>
                   put("        Exceeded the channel capacity in queue_21 at state: ");
                   put(parent_Gstate.system_state_number, width => 1);
                   new_line;
                   put(reach, "        Exceeded the channel capacity in queue_21 at state: ");
                   put(reach, parent_Gstate.system_state_number, width => 1);
                   new_line(reach);
                   place_holder2 := place_holder2.Slink;
                   exit;
             end;
          end loop;
       end loop;
       case search is
          when bfs =>
             if (IsEmpty(Gpointer_queue)) then
                exit;
             else
                parent_Gstate := First(Gpointer_queue);
                Dequeue(Gpointer_queue);
                M1_state := parent_Gstate.Gtuple.machine1_state;
                M2_state := parent_Gstate.Gtuple.machine2_state;
             end if;
          when dfs =>
             if (IsEmpty(Gpointer_stack)) then
                exit;
             else
                Pop(Gpointer_stack, parent_Gstate);
                M1_state := parent_Gstate.Gtuple.machine1_state;
                M2_state := parent_Gstate.Gtuple.machine2_state;
             end if;
          when others =>
             put_line("error error error");
       end case;
    end loop main_loop;
 new_line;
```

```
    number_of_nodes := Gstate_counter;
    put("       Number of nodes in the Reachability Graph: ");
    put(Gstate_counter, width => 1);
    new_line;
end build_Gstate_graph;
```

```
separate(main)
--=============================================================================================
-- Title       : clear_pointers
-- File        : reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Clear all pointers to apply to another user input file.
--=============================================================================================

procedure clear_pointers is
begin
   initial_global_state := null;
   for i in 1 .. 2 loop
      for j in 1 .. 10 loop
         machine_array(i).(j) := null;
      end loop;
   end loop;
end clear_pointers;
```

```
separate(main)
--=============================================================================================
-- Title       : search_for_tuple
-- File        : search.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : A function that searchs the reachability graph for a duplicated tuple.  Returns a pointer
--               value of the found tuple location, else returns null.
--=============================================================================================


function search_for_tuple(top : in Glink_type; tuple : in Gstate_record_type) return Glink_type is
   temp : Glink_type := null;
begin
   if (top = null) then
      return null;
   elsif (IsEqual(tuple, top.Gtuple)) then
      return top;
   else
      if ((temp = null) and then (top.link1.new_node = true)) then
         temp := search_for_tuple(top.link1.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link2.new_node = true)) then
         temp := search_for_tuple(top.link2.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link3.new_node = true)) then
         temp := search_for_tuple(top.link3.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link4.new_node = true)) then
         temp := search_for_tuple(top.link4.Glink, tuple);
      end if;
      return temp;
   end if;
end search_for_tuple;
```

```
separate(main)
--=============================================================================================
-- Title       : pause_output
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain,  U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Procedure called when default output screen reaches max.  Default max is 25 lines.
--=============================================================================================

procedure pause_output is
   keyboard : string(1 .. 1);
   len      : natural;
begin
   new_line;
   put_line("                                    ENTER to continue");
   get_line(keyboard, len);
   put(ASCII.ESC);
   put_line("[2J");
end pause_output;
```



```
separate(main)
--=============================================================================================
-- Title       : unprompted_pause
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Procedure called when default output screen reaches max.  Default max is 25 lines.
--               User is not prompted for ENTER.
--=============================================================================================
procedure unprompted_pause is
   keyboard : string(1 .. 1);
   len      : natural;
begin
   new_line;
   get_line(keyboard, len);
end unprompted_pause;
```

```
separate(main)
--=============================================================================================
-- Title       : PrintQueue
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Takes as input a queue of characters and prints to default output.
--=============================================================================================
procedure PrintQueue(Q : in queue_pack.Queue) is
begin
   if (Q.tail = 0) then
      put('E');
   else
      for i in 1 .. Q.tail loop
         put(Q.Store(i));
      end loop;
   end if;
end PrintQueue;
```

```
separate(main)
--=============================================================================================================-
-- Title       : output_Gstate_node
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs node information to default output.  Displays current state of each machine,
--               the contents of each queue, and the global tuple number.  If the link to another node
--               is null an error flag is set and the link information (transition information) is not
--               output. If both message queues are empty and there are no outgoing transitions from
--               the node then output an unspecified reception condition.  If there are no outgoing
--               transitions and either of the queues are not empty output that an unspecified reception
--               has occured.
--=============================================================================================================-
procedure output_Gstate_node(Gstate_pointer : in out Glink_type; Error_flag : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      unprompted_pause;
   end if;
   set_col(15);
   put(Gstate_pointer.system_state_number, width => 3);
   put("   [" & integer'image(Gstate_pointer.Gtuple.machine1_state) & " , ");
   PrintQueue(Gstate_pointer.Gtuple.queue_12);
   put(" , ");
   PrintQueue(Gstate_pointer.Gtuple.queue_21);
   put(" ," & integer'image(Gstate_pointer.Gtuple.machine2_state) & " ]  ");
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null) and then
    (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null)) then
      if ((Gstate_pointer.Gtuple.queue_12.tail /= 0) or (Gstate_pointer.Gtuple.queue_21.tail /= 0)) then
         set_col(42);
         put(" ******* Unspecified Reception ****** ");
         Error_flag := true;
      else
         set_col(42);
         put(" ******* DEADLOCK Condition ********* ");
         Error_flag := true;
      end if;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_Gstate_transition    .
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs global state transition to default output.  Accepts a pointer to the global
--               state graph as input. The output contains transition type and message.
--=============================================================================================================

procedure output_Gstate_transition(Gstate_pointer : in out Glink_type) is
begin
   if (Gstate_pointer.link1.Glink /= null) then
      set_col(42);
      if (Gstate_pointer.link1.Gtransition = snd) then
         put("-");
      else
         put("+");
      end if;
      put(Gstate_pointer.link1.Gmessage);
      put("  ");
      put("[" & integer'image(Gstate_pointer.link1.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue(Gstate_pointer.link1.Glink.Gtuple.queue_12);
      put(" , ");
      PrintQueue(Gstate_pointer.link1.Glink.Gtuple.queue_21);
      put(" ," & integer'image(Gstate_pointer.link1.Glink.Gtuple.machine2_state) & " ]  ");
      put(Gstate_pointer.link1.Glink.system_state_number, width => 4);
      new_line;
   end if;
   if (Gstate_pointer.link2.Glink /= null) then
      set_col(42);
      if (Gstate_pointer.link2.Gtransition = snd) then
         put("-");
      else
         put("+");
      end if;
      put(Gstate_pointer.link2.Gmessage);
      put("  ");
      put("[" & integer'image(Gstate_pointer.link2.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue(Gstate_pointer.link2.Glink.Gtuple.queue_12);
      put(" , ");
      PrintQueue(Gstate_pointer.link2.Glink.Gtuple.queue_21);
      put(" ," & integer'image(Gstate_pointer.link2.Glink.Gtuple.machine2_state) & " ]  ");
      put(Gstate_pointer.link2.Glink.system_state_number, width => 4);
```

```
      new_line;
   end if;
   if (Gstate_pointer.link3.Glink /= null) then
      set_col(42);
      if (Gstate_pointer.link3.Gtransition = snd) then
         put("-");
      else
         put("+");
      end if;
      put(Gstate_pointer.link3.Gmessage);
      put("  ");
      put("[" & integer'image(Gstate_pointer.link3.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue(Gstate_pointer.link3.Glink.Gtuple.queue_12);
      put(" , ");
      PrintQueue(Gstate_pointer.link3.Glink.Gtuple.queue_21);
      put(" ," & integer'image(Gstate_pointer.link3.Glink.Gtuple.machine2_state) & " ]  ");
      put(Gstate_pointer.link3.Glink.system_state_number, width => 4);
      new_line;
   end if;
   if (Gstate_pointer.link4.Glink /= null) then
      set_col(42);
      if (Gstate_pointer.link4.Gtransition = snd) then
         put("-");
      else
         put("+");
      end if;
      put(Gstate_pointer.link4.Gmessage);
      put("  ");
      put("[" & integer'image(Gstate_pointer.link4.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue(Gstate_pointer.link4.Glink.Gtuple.queue_12);
      put(" , ");
      PrintQueue(Gstate_pointer.link4.Glink.Gtuple.queue_21);
      put(" ," & integer'image(Gstate_pointer.link4.Glink.Gtuple.machine2_state) & " ]  ");
      put(Gstate_pointer.link4.Glink.system_state_number, width => 4);
      new_line;
   end if;

end output_Gstate_transition;
```

```
separate(main)
--=============================================================================================
-- Title       : output_Gstates
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs the entire reachability graph.  Calls procedures output_Gstates and output_
--               Gstate_node.  Accepts pointer to top global state and number of nodes in the graph
--               as variables. Output is directed to default ouput (CRT).
--=============================================================================================

procedure output_Gstates(top_Gstate : in out Glink_type; number_of_nodes : in out natural) is
   Error_flag : boolean    := true;
   current    : Glink_type := top_Gstate;
begin
   if (top_Gstate = null) then
      return;
   else
      output_Gstate_node(top_Gstate, Error_flag);
      if (not Error_Flag) then
         output_Gstate_transition(top_Gstate);
      end if;
      if (top_Gstate.link1.new_node = true) then
         output_Gstates(top_Gstate.link1.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link2.new_node = true) then
         output_Gstates(top_Gstate.link2.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link3.new_node = true) then
         output_Gstates(top_Gstate.link3.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link4.new_node = true) then
         output_Gstates(top_Gstate.link4.Glink, number_of_nodes);
      end if;
   end if;
end output_Gstates;
```

```
separate(main)
-------------------------------------------------------------------------------------------------
-- Title       : output_machine_arrays
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Output the contents of each machine array to an output text file.  Accepts a pointer
--               to a three dimensional machine array (pointers to machine arrays).
-------------------------------------------------------------------------------------------------

procedure output_machine_arrays(machine_array : in out system_array_type) is
   place_holder : Slink_type;
   line_count   : integer := 7;
begin
   new_line(2);
   set_col(20);
   put("-----------------------------------------");
   set_col(20);
   put("|      Machine 1 Array Contents          |");
   set_col(20);
   put("-----------------------------------------");
   set_col(20);
   put("| From  |  To  |  Transition  | Executed |");
   set_col(20);
   put("-----------------------------------------");

   for curr_state1 in 1 .. 10 loop
      place_holder := machine_array(1) (curr_state1);
      while (place_holder /= null) loop
         line_count := line_count + 1;
         if ((line_count mod 20) = 0) then
            pause_output;
         end if;
         set_col(20);
         put("|  " & integer'image(curr_state1));
         put("    |  " & integer'image(place_holder.next_Mstate) & "  |      ");
         put(place_holder.transition, set => lower_case);
         put("  " & place_holder.message & "    | ");
         put("  ");
         put(place_holder.visited, set => lower_case, width => 3);
         put("    |");
         new_line;
         place_holder := place_holder.Slink;
      end loop;
   end loop;
   set_col(20);
   put("-----------------------------------------");
   new_line(2);
   set_col(20);
```

```
   put("-----------------------------------------");
   set_col(20);
   put("|       Machine 2 Array Contents         |");
   set_col(20);
   put("-----------------------------------------");
   set_col(20);
   put("| From  |  To  |  Transition  | Executed |");
   set_col(20);
   put("-----------------------------------------");
   for curr_state2 in 1 .. 10 loop
      place_holder := machine_array(2) (curr_state2);
      while (place_holder /= null) loop
         line_count := line_count + 1;
         if ((line_count mod 20) = 0) then
            pause_output;
         end if;
         set_col(20);
         put("|   " & integer'image(curr_state2));
         put("    |   " & integer'image(place_holder.next_Mstate) & "   |      ");
         put(place_holder.transition, set => lower_case);
         put("   " & place_holder.message & "    | ");
         put("  ");
         put(place_holder.visited, set => lower_case, width => 3);
         put("     |");
         new_line;
         place_holder := place_holder.Slink;
      end loop;
   end loop;

   set_col(20);
   put("-----------------------------------------");
   new_line;
end output_machine_arrays;
```

```
separate(main)
--=============================================================================================================
-- Title       : PrintQueue_to_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Prints queue of characters to output file named by the user.
--=============================================================================================================
procedure PrintQueue_to_file(Q : in queue_pack.Queue) is
begin
   if (Q.tail = 0) then
      put(reach, 'E');
   else
      for i in 1 .. Q.tail loop
         put(reach, Q.Store(i));
      end loop;
   end if;
end PrintQueue_to_file;
```

```
separate(main)
--=============================================================================================-
-- Title       : output_Gstate_node_to_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs node information to a text file.  Displays current state of each machine, the
--               contents of each queue, and the global tuple number.  If the link to another node is
--               null an error flag is set and the link information (transition information) is not output.
--               If both message queues are empty and there are no outgoing transitions from the node
--               then output an unspecified reception condition.  If there are no outgoing transitions
--              and either of the queues are not empty output that an unspecified reception has occured.
--=============================================================================================-
procedure output_Gstate_node_to_file(Gstate_pointer : in out Glink_type; Error_flag : in out boolean) is
begin

   set_col(reach, 15);
   put(reach, Gstate_pointer.system_state_number, width => 3);
   put(reach, "   [" & integer'image(Gstate_pointer.Gtuple.machine1_state) & " , ");
   PrintQueue_to_file(Gstate_pointer.Gtuple.queue_12);
   put(reach, " , ");
   PrintQueue_to_file(Gstate_pointer.Gtuple.queue_21);
   put(reach," ," & integer'image(Gstate_pointer.Gtuple.machine2_state) & " ]  ");
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null) and then
    (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null)) then
      if ((Gstate_pointer.Gtuple.queue_12.tail /= 0) or (Gstate_pointer.Gtuple.queue_21.tail /= 0)) then
         set_col(reach, 42);
         put(reach, " ******* Unspecified Reception ****** ");
         Error_flag := true;
      else
         set_col(reach, 42);
         put(reach, " ******* DEADLOCK Condition ********* ");
         Error_flag := true;
      end if;
   else
      Error_flag := false;
   end if;
end output_Gstate_node_to_file;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_Gstate_transition_to_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs global state transition to an output file.  Accepts a pointer to the global
--               state graph as input. The output contains transition type and message.
--=============================================================================================================

procedure output_Gstate_transition_to_file(Gstate_pointer : in out Glink_type) is
begin
   if (Gstate_pointer.link1.Glink /= null) then
      set_col(reach, 42);
      if (Gstate_pointer.link1.Gtransition = snd) then
         put(reach, "-");
      else
         put(reach, "+");
      end if;
      put(reach, Gstate_pointer.link1.Gmessage);
      put(reach, "  ");
      put(reach, "[" & integer'image(Gstate_pointer.link1.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue_to_file(Gstate_pointer.link1.Glink.Gtuple.queue_12);
      put(reach, " , ");
      PrintQueue_to_file(Gstate_pointer.link1.Glink.Gtuple.queue_21);
      put(reach, " ," & integer'image(Gstate_pointer.link1.Glink.Gtuple.machine2_state) & " ]  ");
      put(reach, Gstate_pointer.link1.Glink.system_state_number, width => 4);
      new_line(reach);
   end if;
   if (Gstate_pointer.link2.Glink /= null) then
      set_col(reach, 42);
      if (Gstate_pointer.link2.Gtransition = snd) then
         put(reach, "-");
      else
         put(reach, "+");
      end if;
      put(reach, Gstate_pointer.link2.Gmessage);
      put(reach, "  ");
      put(reach, "[" & integer'image(Gstate_pointer.link2.Glink.Gtuple.machine1_state) & " , ");
      PrintQueue_to_file(Gstate_pointer.link2.Glink.Gtuple.queue_12);
      put(reach, " , ");
      PrintQueue_to_file(Gstate_pointer.link2.Glink.Gtuple.queue_21);
      put(reach, " ," & integer'image(Gstate_pointer.link2.Glink.Gtuple.machine2_state) & " ]  ");
      put(reach, Gstate_pointer.link2.Glink.system_state_number, width => 4);
      new_line(reach);
   end if;
   if (Gstate_pointer.link3.Glink /= null) then
      set_col(reach, 42);
      if (Gstate_pointer.link3.Gtransition = snd) then
```

```
            put(reach, "-");
         else
            put(reach, "+");
         end if;
         put(reach, Gstate_pointer.link3.Gmessage);
         put(reach, "  ");
         put(reach, "[" & integer'image(Gstate_pointer.link3.Glink.Gtuple.machine1_state) & " , ");
         PrintQueue_to_file(Gstate_pointer.link3.Glink.Gtuple.queue_12);
         put(reach, " , ");
         PrintQueue_to_file(Gstate_pointer.link3.Glink.Gtuple.queue_21);
         put(reach, " ," & integer'image(Gstate_pointer.link3.Glink.Gtuple.machine2_state) & " ]  ");
         put(reach, Gstate_pointer.link3.Glink.system_state_number, width => 4);
         new_line(reach);
      end if;
      if (Gstate_pointer.link4.Glink /= null) then
         set_col(reach, 42);
         if (Gstate_pointer.link4.Gtransition = snd) then
            put(reach, "-");
         else
            put(reach, "+");
         end if;
         put(reach, Gstate_pointer.link4.Gmessage);
         put(reach, "  ");
         put(reach, "[" & integer'image(Gstate_pointer.link4.Glink.Gtuple.machine1_state) & " , ");
         PrintQueue_to_file(Gstate_pointer.link4.Glink.Gtuple.queue_12);
         put(reach, " , ");
         PrintQueue_to_file(Gstate_pointer.link4.Glink.Gtuple.queue_21);
         put(reach, " ," & integer'image(Gstate_pointer.link4.Glink.Gtuple.machine2_state) & " ]  ");
         put(reach, Gstate_pointer.link4.Glink.system_state_number, width => 4);
         new_line(reach);
      end if;

   end output_Gstate_transition_to_file;
```

```
separate(main)
-- ============================================================================================================
-- Title       : output_Gstates_to_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs the entire reachability graph.  Calls procedures output_Gstates and output_
--               Gstate_node.  Accepts pointer to top global state and number of nodes in the graph
--               as variables. Output is directed to a text output file.
-- ============================================================================================================

procedure output_Gstates_to_file(top_Gstate : in out Glink_type; number_of_nodes : in out natural) is
   Error_flag : boolean    := true;
   current    : Glink_type := top_Gstate;
begin
   if (top_Gstate = null) then
      return;
   else
      output_Gstate_node_to_file(top_Gstate, Error_flag);
      if (not Error_Flag) then
         output_Gstate_transition_to_file(top_Gstate);
      end if;
      if (top_Gstate.link1.new_node = true) then
         output_Gstates_to_file(top_Gstate.link1.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link2.new_node = true) then
         output_Gstates_to_file(top_Gstate.link2.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link3.new_node = true) then
         output_Gstates_to_file(top_Gstate.link3.Glink, number_of_nodes);
      end if;
      if (top_Gstate.link4.new_node = true) then
         output_Gstates_to_file(top_Gstate.link4.Glink, number_of_nodes);
      end if;
   end if;
end output_Gstates_to_file;
```

```
separate(main)
--=============================================================================================
-- Title       : output_machine_arrays_to_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Output the contents of each machine array to default output (CRT).  Accepts a pointer
--               to a three dimensional machine array (pointers to machine arrays).
--=============================================================================================

procedure output_machine_arrays_to_file(machine_array : in out system_array_type) is
   place_holder : Slink_type;
begin
   new_line(reach, 2);
   set_col(reach, 20);
   put(reach, "-----------------------------------------");
   set_col(reach, 20);
   put(reach, "|      Machine 1 Array Contents          |");
   set_col(reach, 20);
   put(reach, "-----------------------------------------");
   set_col(reach, 20);
   put(reach, "| From  |  To  |  Transition  | Executed |");
   set_col(reach, 20);
   put(reach, "-----------------------------------------");

   for curr_statel in 1 .. 10 loop
      place_holder := machine_array(1) (curr_statel);
      while (place_holder /= null) loop
         set_col(reach, 20);
         put(reach, "|  " & integer'image(curr_statel));
         put(reach, "    |  " & integer'image(place_holder.next_Mstate) & "  |      ");
         put(reach, place_holder.transition, set => lower_case);
         put(reach, "  " & place_holder.message & "    | ");
         put(reach, "  ");
         put(reach, place_holder.visited, set => lower_case, width => 3);
         put(reach, "     |");
         new_line(reach);
         place_holder := place_holder.Slink;
      end loop;
   end loop;
   set_col(reach, 20);
   put(reach, "-----------------------------------------");
   new_line(reach, 2);
   set_col(reach, 20);
```

```
      put(reach, "-----------------------------------------");
      set_col(reach, 20);
      put(reach, "|       Machine 2 Array Contents         |");
      set_col(reach, 20);
      put(reach, "-----------------------------------------");
      set_col(reach, 20);
      put(reach, "| From  |  To  |  Transition  | Executed |");
      set_col(reach, 20);
      put(reach, "-----------------------------------------");
      for curr_state2 in 1 .. 10 loop
         place_holder := machine_array(2) (curr_state2);
         while (place_holder /= null) loop
            set_col(reach, 20);
            put(reach, "|  " & integer'image(curr_state2));
            put(reach, "    |  " & integer'image(place_holder.next_Mstate) & "  |      ");
            put(reach, place_holder.transition, set => lower_case);
            put(reach, "  " & place_holder.message & "    | ");
            put(reach, "  ");
            put(reach, place_holder.visited, set => lower_case, width => 3);
            put(reach, "     |");
            new_line(reach);
            place_holder := place_holder.Slink;
         end loop;
      end loop;

      set_col(reach, 20);
      put(reach, "-----------------------------------------");
      new_line(reach, 2);
      put(reach, "                        * The nodes generated by the analysis");
      new_line(reach);
      put(reach, "                          were done in a ");
      case search is
         when dfs =>
            put_line(reach, "depth first manner");
         when bfs =>
            put_line(reach, "breadth first manner");
         when others =>
            put_line("error");
      end case;
      new_line(reach);
   end output_machine_arrays_to_file;
```

```
separate(main)
--=============================================================================================
-- Title       : create_output_file
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Creates an output file that the user names and places appropriate header information
--               at the beginning of the text file.
--=============================================================================================

procedure create_output_file is
begin
   set_col(10);
   put("Filename of output analysis: ");
   get_line(out_file_name, out_name_len);
   create(reach, mode => out_file, name => out_file_name(1 .. out_name_len));
   put(ASCII.ESC);
   put_line("[2J");
   new_line(reach, 2);
   put(reach, "                              REACHABILITY ANALYSIS of : " & out_file_name(1 .. out_name_len));
   new_line(reach, 2);
end create_output_file;
```

```
separate(main)
-------------------------------------------------------------------------------------------------
-- Title       : output_analysis
-- File        : output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure builds the machine arrays, and produces the global reachability graph.
--               Parameters are described in the header of each procedure.
-------------------------------------------------------------------------------------------------

procedure output_analysis is
begin
   pause_output;
   output_machine_arrays(machine_array);
   pause_output;
   build_Gstate_graph(machine_array, start_state_1, start_state_2, initial_global_state, number_of_nodes);
   pause_output;
   new_line(2);
   put_line("                                REACHABILITY GRAPH    ");
   new_line;
   output_line_count := 1;
   output_Gstates(initial_global_state, number_of_nodes);
   pause_output;
   output_machine_arrays(machine_array);
   pause_output;
   new_line(2);
   new_line(reach, 2);
   put_line(reach, "                                Reachability Graph");
   output_Gstates_to_file(initial_global_state, number_of_nodes);
   new_line(reach);
   output_machine_arrays_to_file(machine_array);
end output_analysis;
```

APPENDIX B SCM CODE

The program listing begins on the following page.

```
--=============================================================================================================
-- Title       : main
-- File        : main.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 16 March 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This is the main procedure.  The user provides a text file that is the specification of a two
--               machine model.  The text file is parsed and machine adjacency lists are built.  A global and
--               system reachability analysis is done on the machine specification and output to screen and
--               text file. All procedures that the main uses were created and compiled as separate units.
--               Desriptions of each function and procedure can be found in the header of  the associated
--               units/files.
--=============================================================================================================


with Text_IO, queues, stacks, definitions;
use  Text_IO, definitions;

--#########################################################
--########### User Defined: package definitions   #######
--###########          found in file user_spec.a  #######
--#########################################################

procedure main is

   max_length : natural := 40;                    --length of global state array
   type scm_instruction_type is (start, number_of_machines, machine, number_of_states, state, initial_state, trans,
    finish);

   type visit_type            is (yes, no);

   package trans_enum_io is new Text_IO.Enumeration_IO(scm_transition_type);
   package instr_enum_io is new Text_IO.Enumeration_IO(scm_instruction_type);
   package vst_enum_io is new Text_IO.Enumeration_IO(visit_type);
   package buff_enum_io is new Text_IO.Enumeration_IO(buffer_type);
   package scm_integer_io is new Integer_IO(integer);
   package bool_enum_io is new Text_IO.Enumeration_IO(boolean_type);
   use bool_enum_io;

   use  trans_enum_io, instr_enum_io, vst_enum_io,
        buff_enum_io, scm_integer_io;
   ------------------------------------------------------------------------
   -- data structures for the machine arrays
   ------------------------------------------------------------------------
   type machine_array_record_type;
   type Slink_type is access machine_array_record_type;
   type machine_array_record_type is
      record
         transition     : scm_transition_type := unused;
         next_Mstate    : natural    := 0;
         visited        : visit_type := no;
         Slink          : Slink_type := null;
   end record;
```

```
type machine_array_type is array(integer range 0 .. 10) of Slink_type;

type system_array_type is array(1 .. 2) of machine_array_type;

------------------------------------------------------------------------
--data structures for the global state tuple(node)
------------------------------------------------------------------------
type global_state_type;
type Glink_type is access global_state_type;

------------------------------------------------------------------------
--transition structure
------------------------------------------------------------------------
type Gstate_transition_type is
   record
      Gtransition : scm_transition_type := unused;
      new_node    : boolean             := true;
      Glink       : Glink_type;
      visited     : boolean             := false;
end record;

------------------------------------------------------------------------
--global tuple structure
------------------------------------------------------------------------
type Gstate_record_type is
   record
      machine1_state   : machine1_state_type;
      machine2_state   : machine2_state_type;
      global_variables : global_variable_type;
 end record;

------------------------------------------------------------------------
--Global state node, contains transition, state, and link information
--needed for building the global state graph
------------------------------------------------------------------------
type global_state_type is
   record
      system_state_number : natural := 0;
      Gtuple : Gstate_record_type;
      link1  : Gstate_transition_type;
      link2  : Gstate_transition_type;
      link3  : Gstate_transition_type;
      link4  : Gstate_transition_type;
end record;

type system_state_type;
type Syslink_type is access system_state_type;
```

```
--------------------------------------------------------------------------
--transition structure for system state
--------------------------------------------------------------------------
type Sstate_transition_type is
   record
      Stransition : scm_transition_type := unused;
      new_node    : boolean             := true;
      Syslink     : Syslink_type;
end record;

type Sstate_record_type is
   record
      machine1_state   : natural := 0;
      machine2_state   : natural := 0;
 end record;

--------------------------------------------------------------------------
--system state structure
--------------------------------------------------------------------------
type system_state_type is
   record
      system_state_number : natural := 0;
      Stuple : Sstate_record_type;
      link1  : Sstate_transition_type;
      link2  : Sstate_transition_type;
      link3  : Sstate_transition_type;
      link4  : Sstate_transition_type;
end record;

--------------------------------------------------------------------------
--declaration of stacks of pointers and queues of pointers
--along with queues of messages to simulate two way communications
--between machines
--------------------------------------------------------------------------
package transition_stack_package is new stacks(scm_transition_type,    MAX => 5);
package Gtrans_stack_package     is new stacks(Gstate_transition_type, MAX => 50);
package Gtrans_queue_package     is new queues(Gstate_transition_type, MAX => 50);
package Strans_queue_package     is new queues(Sstate_transition_type, MAX => 50);
package Strno_queue_package      is new queues(integer,                MAX => 50);
package Gpointer_stack_package   is new stacks(Glink_type,             MAX => 50);
package Gpointer_queue_package   is new queues(Glink_type,             MAX => 50);
package Spointer_queue_package   is new queues(Syslink_type,          MAX => 50);
package Gstate_stack_package     is new stacks(global_state_type);
package count_inout              is new integer_io(count);
 use  Gpointer_stack_package, Gstate_stack_package, Gpointer_queue_package,
     transition_stack_package,count_inout,Gtrans_stack_package,Gtrans_queue_package,
     Strans_queue_package,Strno_queue_package, Spointer_queue_package;


--------------------------------------------------------------------------
--global variables are initialized
--------------------------------------------------------------------------
initial_global_state : Glink_type;              -- top tuple in global graph
initial_system_state : Syslink_type;
```

```
scm_instruction      : scm_instruction_type;   --used in case statements
total_machines       : natural := 0;           --number of machines in the system
total_states         : natural := 0;           --number of states per machine
current_state        : natural := 0;           --current_state of the current machine
next_state           : natural := 0;
current_machine      : natural := 0;
current_transition   : scm_transition_type;
current_message      : character;              --message off the top of the queue
transition_stack     : transition_stack_package.stack;
start_state_1        : natural := 0;
start_state_2        : natural := 0;
temp_Gstate          : Gstate_record_type;     -- a temporary holding place for tuples
machine_array        : system_array_type;      -- an array of arrays so that the machines can be subscripted to be
Gpointer_stack       : Gpointer_stack_package.stack;
Gpointer_queue       : Gpointer_queue_package.queue;
Strans_queue         : Strans_queue_package.queue;
link_queue           : Strno_queue_package.queue;
Gtrans_stack         : Gtrans_stack_package.stack;
Gtrans_queue         : Gtrans_queue_package.queue;
Sstate               : Spointer_queue_package.queue;
number_of_nodes      : natural := 0;
output_line_count    : natural := 1;           --stops output from scrolling if longer than 25 lines
file_name            : string(1 .. 20) := (others => ' ');
out_file_name        : string(1 .. 20) := (others => ' ');
in_name_len          : integer;                --variables that holds length of a string
out_name_len         : integer;
analysis_len         : integer;
transit_len          : integer := 8;
Scolumn_set          : positive_count := 35;
Gcolumn_set          : positive_count := 5;
scm                  : file_type;              --input file pointer
reach                : file_type;              --output file pointer

-------------------------------------------------------------------------------------------------------------------
procedure scroll_pause                                          is separate;                    --global_output.a
procedure output_Stuple                  (tuple          : in out Sstate_record_type)       --system_output.a
                                          is separate;                                      --
procedure output_Sstate_node             (Sstate_pointer : in out Syslink_type;             --system_output.a
                                          Error_flag     : in out boolean) is separate;     --
procedure output_Sstate_transition       (Sstate_pointer : in out Syslink_type)             --system_output.a
                                          is separate;                                      --
procedure output_Sstates                 (top_Sstate     : in out Syslink_type)             --system_output.a
                                          is separate;                                      --
procedure clear_pause                                           is separate;                    --global_output.a

--################################################## User Defined ##################################################
procedure output_Gtuple                  (tuple          : in out Gstate_record_type)       --                #
                                          is separate;                                      --user_output.a #
procedure output_Gtuple_to_file          (tuple          : in out Gstate_record_type;       --user_output.a #
```

```
                                        counter        : in out integer) is separate;      --              #
   --#####################################################################################################

   procedure output_Gstate_node         (Gstate_pointer : in out Glink_type;                --global_output.a
                                         Error_flag     : in out boolean) is separate;      --
   procedure output_Gstate_transition   (Gstate_pointer : in out Glink_type) is separate;   --global_output.a
   procedure output_Gstates             (top_Gstate     : in out Glink_type) is separate;   --global_output.a
   procedure output_machine_arrays      (machine_array  : in out system_array_type)         --global_output.a
                                         is separate;                                       --
   procedure load_machine_array         (number         : in out natural;                   --input.a
                                         state          : in out natural;                   --
                                         Next_state     : in out natural;                   --
                                         Transition     : in out scm_transition_type)       --
                                         is separate;                                       --
   procedure read_in_file               (scm            : in out file_type) is separate;    --input.a
   function IsSysStateEqual             (top            : in Syslink_type;                  --system_reachability.a
                                         current        : in system_state_type)             --
                                         return boolean is separate;                        --
   function search_for_Gtuple           (top            : in Glink_type;                    --global_search.a
                                         tuple          : in Gstate_record_type)            --
                                         return Glink_type is separate;                     --
   function search_for_Stuple           (top            : in Syslink_type;                  --system_search.a
                                         current : in system_state_type)                    --
                                         return Syslink_type is separate;                   --

   --################################################## User Defined ###################################################
   function  Analyze_Predicates_Machine1 (local         : machine1_state_type;              --predicate_action.a  #
                                         GLOBAL         : global_variable_type)             --                    #
                                         return transition_stack_package.stack              --                    #
                                                  is separate;                              --                    #
   function Analyze_Predicates_Machine2 (local          : machine2_state_type;              --predicate_action.a  #
                                                                GLOBAL        : global_variable_type)
-                  #
                                         return transition_stack_package.stack              --                    #
                                                  is separate;                              --                    #
   procedure Action                     (in_system_state: in out Gstate_record_type;        --predicate_action.a  #
                                         in_transition  : in out scm_transition_type;       --                    #
                                         out_system_state : in out Gstate_record_type)      --                    #
                                         is separate;                                       --                    #
   --#####################################################################################################

   procedure build_Gstate_graph         (machine_array  : in out system_array_type;         --global_reachability.a
                                         start_state_1  : in out natural;                   --
                                         start_state_2  : in out natural;                   --
                                         top_Gstate     : in out Glink_type) is separate;   --
   procedure build_Sstate_graph         (top_Gstate     : in out Glink_type;                --system_reachability.a
                                         top_Sstate     : in out Syslink_type) is separate; --
   procedure output_analysis                              is separate;                      --global_output.a
   procedure clear_pointers                               is separate;                      --global_reachability.a
   ---------------------------------------------------------------------------------------------------------------


begin
   main_scm_loop :
```

```
      loop
         put(ASCII.ESC);     --call to operating system to clear the screen
         put_line("[2J");
         new_line(10);

         --query the user for the input file
         loop
            begin
               set_col(10);
               put("Filename of scm specification (ENTER to quit): ");
               get_line(file_name, in_name_len);
               if (in_name_len = 0) then
                  exit main_scm_loop;
               else
                  open(scm, mode => in_file, name => file_name(1 .. in_name_len));
               end if;
               exit;
            exception
               when NAME_ERROR =>
                  put_line("That file does not exist. Reenter!!");
            end;
         end loop;
         read_in_file(scm);
         clear_pause;
         output_analysis;
         close(scm);
         clear_pointers;
      end loop main_scm_loop;
end main;
```

```
separate(main)
--=============================================================================================
-- Title       : load_machine_array
-- File        : input.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Builds machine adjacency list.  Takes transition and message as input and adds them to
--               the lists.  The possible transitions are determined by the file "user_spec.a".
--               The machine arrays are represented by a single one dimensional array of arrays of
--               linked lists.
--=============================================================================================

procedure load_machine_array(number     : in out natural;
                             state      : in out natural;
                             Next_state : in out natural;
                             Transition : in out scm_transition_type) is

   end_of_list : Slink_type := machine_array(number) (state);
   temp_link   : Slink_type := new machine_array_record_type;
begin

   temp_link.transition := Transition;
   temp_link.next_Mstate := Next_state;
   if (end_of_list = null) then
      machine_array(number) (state) := temp_link;
   else
      while (end_of_list.Slink /= null) loop
         end_of_list := end_of_list.Slink;
      end loop;
      end_of_list.Slink := temp_link;
   end if;
end load_machine_array;
```

```
separate(main)
--=============================================================================================================
-- Title       : read_in_file
-- File        : input.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Parses input file one line at a time. Each line is read into a line buffer and tokens
--               are extracted from the line.  A token may be an understandable machine instruction:
--
--                 start:              placed as a flag at the beginning of the file
--                 number_of_states:   a variable used to tell the dimensions of
--                                     machine array.
--                 number_of_machines: same as above, this variable is not used
--                                       but built in to use for more dynamic versions
--                 machine:            a flag to identify the array that the machine
--                                     instructions are to be loaded.
--                 state:              identifies the row of the machine array that
--                                     the instruction and message are to be loaded
--                 trans:              identify a transition <trans -X 2> means
--                                     transition from the current machine and
--                                     state that is a send with message X to
--                                     state 2.
--                 intial_state:       identifies the starting state of machines.
--                                     <initial_state 1 1> means that the starting
--                                     state of machine one is state one and the
--                                     starting state of machine two is state one.
--                 finish:             a end of text file delimeter.
--
--               As the file is scanned and the text is converted to usable tokens the tokens
--               are output to default output(CRT) for the user to view. The tokens determine
--               how the machine adjacency lists are built.
--
--=============================================================================================================

procedure read_in_file(scm : in out file_type) is

   line_buffer : string(1 .. 80) := (others => ' ');
   token       : string(1 .. 80) := (others => ' ');
   token_index : integer         := 1;
   line_length : integer;
   line_index  : integer         := 1; --variable used for halting output every
   enum_length : integer;              --25 lines.
   conv_length : integer;
   line_count  : integer         := 1;

   --clears the token used in parsing
```

```
   procedure clear_token(in_token : in out string) is
   begin
      for i in 1 .. 80 loop
         in_token(i) := ' ';
      end loop;
   end clear_token;

begin
   new_line(4);
   while not End_of_File(scm) loop
      line_count := line_count + 1;
      if ((line_count mod 20) = 0) then
         scroll_pause;
      end if;
      get_line(scm, line_buffer, line_length);

      while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
         line_index := line_index + 1;
      end loop;

      while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
         token(token_index) := line_buffer(line_index);
         token_index := token_index + 1;
         line_index := line_index + 1;
      end loop;

      instr_enum_io.get(token(1 .. token_index - 1), scm_instruction, enum_length);
      token_index := 1;
      clear_token(token);
      case scm_instruction is
         when start =>
            put_line("            PROGRAM IS LOADING.....");
            new_line(2);
            set_col(20);
            put(scm_instruction, set => lower_case);
            line_index := 1;

         when number_of_machines =>
            set_col(20);
            put(scm_instruction, set => lower_case);
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;

            get(token, total_machines, conv_length);
            put(total_machines, width => 2);
            line_index := 1;
            token_index := 1;
            clear_token(token);
```

```
when number_of_states =>
   set_col(20);
   put(scm_instruction, set => lower_case);
   while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
      line_index := line_index + 1;
   end loop;
   while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
      token(token_index) := line_buffer(line_index);
      token_index := token_index + 1;
      line_index := line_index + 1;
   end loop;

   get(token, total_states, conv_length);
   put(total_states, width => 2);
   line_index := 1;
   token_index := 1;
   clear_token(token);

when machine =>
   set_col(20);
   put(scm_instruction, set => lower_case);
   while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
      line_index := line_index + 1;
   end loop;
   while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
      token(token_index) := line_buffer(line_index);
      token_index := token_index + 1;
      line_index := line_index + 1;
   end loop;

   get(token, current_machine, conv_length);
   put(current_machine, width => 2);
   line_index := 1;
   token_index := 1;
   clear_token(token);

when state =>
   set_col(20);
   put(scm_instruction, set => lower_case);
   while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
      line_index := line_index + 1;
   end loop;
   while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
      token(token_index) := line_buffer(line_index);
      token_index := token_index + 1;
      line_index := line_index + 1;
   end loop;
```

```
            get(token, current_state, conv_length);
            put(current_state, width => 2);
            line_index := 1;
            token_index := 1;
            clear_token(token);

         when trans =>
            set_col(20);
            put(scm_instruction, set => lower_case);
            put(" ");
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;
            trans_enum_io.get(token(1 .. token_index - 1), current_transition, enum_length);
            put(current_transition,set=>lower_case);
            token_index := 1;
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;
            get(token, next_state, conv_length);
            put(next_state, width => 2);
            token_index := 1;
            line_index := 1;
            load_machine_array(current_machine, current_state, next_state, current_transition);

         when initial_state =>
            set_col(20);
            put(scm_instruction, set => lower_case);
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
               token_index := token_index + 1;
               line_index := line_index + 1;
            end loop;
            get(token, start_state_1, conv_length);
            put(start_state_1, width => 2);
            token_index := 1;
            while line_index < line_length + 1 and then line_buffer(line_index) = ' ' loop
               line_index := line_index + 1;
            end loop;
            while line_index < line_length + 1 and then line_buffer(line_index) /= ' ' loop
               token(token_index) := line_buffer(line_index);
```

```
            token_index := token_index + 1;
            line_index := line_index + 1;
         end loop;
         get(token, start_state_2, conv_length);
         put(start_state_2, width => 2);
         line_index := 1;
         token_index := 1;
         clear_token(token);

      when finish =>
         set_col(20);
         put(scm_instruction, set => lower_case);
         new_line(2);
         put_line("       machines's are  loaded!!!");
         exit;

      when others =>
         put_line("                 error---error---error");
         for x in 1 .. line_length loop
            put(line_buffer(x));
         end loop;

   end case;
   new_line;
end loop;
end read_in_file;
```

```
--=============================================================================================
-- Title       : queues
-- File        : queues.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This generic package contains procedures and functions to implement a queue of
--               type item.  In the model there are queues of characters (to simulate communication
--               channels) and queues of global tuple pointers to aid in the construction
--               of the global reachability graph.  The variables chosen for the procedures and functions
--               are self descriptive.  The default size of a queue is 10, the user has the option of
--               making the queue bigger ( up to the maximum size of the supporting hardware.) A number
--               of the functions were taken from the text, "Data Structures in Ada" by Michael Feldman.
--=============================================================================================

generic

   type Item is private;
   MAX : in integer := 10;

package queues is

   type ArrayList is array(1 .. MAX) of Item;
   type Queue is
      record
         Store : ArrayList;
         tail  : integer range 0 .. MAX := 0;
   end record;

   procedure MakeEmpty(Q : in out Queue);
   procedure Enqueue(Q : in out Queue; E : in Item);
   procedure Dequeue(Q : in out Queue);
   function First(Q : Queue) return Item;
   function IsEmpty(Q : Queue) return boolean;
   function IsEqual(Q1 : Queue; Q2 : Queue) return boolean;

   QueueFull  : exception;
   QueueEmpty : exception;

end queues;

package body queues is

   procedure MakeEmpty(Q : in out queue) is
      blank_line : string(1 .. 80) := (others => ' ');
   begin
      Q.tail := 0;
   end MakeEmpty;

   procedure Enqueue(Q : in out Queue; E : in Item) is
   begin
```

```
      if Q.tail = MAX then
         raise QueueFull;
      else
         Q.tail := Q.tail + 1;
         Q.Store(Q.tail) := E;
      end if;
   end Enqueue;

   procedure Dequeue(Q : in out Queue) is
   begin
      if Q.tail = 0 then
         raise QueueEmpty;
      else
         for i in 2 .. Q.tail loop
            Q.Store(i - 1) := Q.Store(i);
         end loop;
         Q.tail := Q.tail - 1;
      end if;
   end Dequeue;


   function First(Q : Queue) return Item is
   begin
      if Q.tail = 0 then
         raise QueueEmpty;
      else
         return Q.Store(1);
      end if;
   end First;

   function IsEmpty(Q : Queue) return boolean is
   begin
      return Q.tail = 0;
   end IsEmpty;

   function IsEqual(Q1 : Queue; Q2 : Queue) return boolean is
   begin
      for i in 1 .. Q1.tail loop
         if (Q1.Store(i) /= Q2.Store(i)) then
            return false;
         end if;
      end loop;
      return true;
   end IsEqual;

end queues;
```

```
--=============================================================================================
-- Title       : stacks
-- File        : stacks.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 February 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This generic package contains procedures and functions to implement a stack of
--               type item.  In the model there are only stacks of pointers to aid in the construction
--               of the global reachability graph.  The variables chosen for the procedures and functions
--               are self descriptive.  The default size of a queue is 10, the user has the option of
--               making the queue bigger ( up to the maximum size of the supporting hardware.) A number
--               of the functions were taken from the text, "Data Structures in Ada" by Michael Feldman.
--=============================================================================================


generic

   type Item is private;
   MAX : in integer := 10;

package stacks is

   type ArrayList is array(1 .. MAX) of Item;
   type Stack is
      record
         Store  : ArrayList;
         latest : integer range 0 .. MAX := 0;
   end record;

   procedure MakeEmpty(S : in out Stack);
   function IsEmpty(S : Stack) return boolean;
   procedure Push(S : in out Stack; E : in Item);
   procedure Pop(S : in out Stack; E : out Item);
   function Top(S : Stack) return Item;

   StackFull  : exception;
   StackEmpty : exception;

end stacks;

package body stacks is

   procedure MakeEmpty(S : in out Stack) is
   begin
      S.latest := 0;
   end MakeEmpty;

   procedure Push(S : in out Stack; E : in Item) is
   begin
      if S.latest = MAX then
         raise StackFull;
      else
```

```
            S.latest := S.latest + 1;
            S.Store(S.latest) := E;
         end if;
      end Push;

      procedure Pop(S : in out Stack; E : out Item) is
      begin
         if S.latest = 0 then
            raise StackEmpty;
         else
            E := S.Store(S.latest);
            S.latest := S.latest - 1;
         end if;
      end Pop;

      function Top(S : Stack) return Item is
      begin
         if S.latest = 0 then
            raise StackEmpty;
         else
            return S.Store(S.latest);
         end if;
      end Top;

      function IsEmpty(S : Stack) return boolean is
      begin
         return S.latest = 0;
      end IsEmpty;

   end stacks;
```

```
separate(main)
--=============================================================================================================
-- Title       : build_Gstate_graph
-- File        : global_reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 18 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure builds the global reachability graph for two machines.  Each global state node
--               can have up to 4 transitions.  The algorithm for this procedure is as follows.  Each machine
--               state is checked for valid transitions to another state. Transitions are valid if the analyze-
--               predicate procedures puts the transitions onto a holding stack.  A temporary tuple is construct-
--               ed to hold the values of each machines current state and all global/local variables. All nodes
--               in the existing graph are checked to see if the values match the temporary tuple values. If a
--               node already exists, a link is made from the current node to the found node. If a match for the
--               temporary tuple is not found a node is created and inserted into the graph. A pointer to the newly
--               added node is added to the pointer queue.  A pointer is retrieved from the queue after a scan is
--               done from each machine array. This procedure terminates whenever there are no transitions out of
--               any machine array states, or the intial state for each machine is reached, or the pointer queue is
--               empty.  As can be demonstrated, SCM reachability graphs can grow without bound. A max pointer queue
--               size is introduced to eliminate unboundedness.
--=============================================================================================================

procedure build_Gstate_graph(machine_array : in out system_array_type;
                             start_state_1 : in out natural;
                             start_state_2 : in out natural;
                             top_Gstate    : in out Glink_type) is
   parent_Gstate, current_Gstate, find_Gstate, temp_tuple_pointer : Glink_type := null;
   Gstate_counter                                                 : integer    := 0;
   M1_state, M2_state                                             : natural;    --machine 1 and machine 2 state holder
   next_state                                                     : natural;    --returned value from a transition
   row_holder1, position_holder1, row_holder2, position_holder2   : Slink_type;
   initial_Gstate                                                 : Gstate_record_type;
   is_new                                                         : boolean;
   toggle                                                         : boolean    := false;
```

```
--=============================================================================================================
-- Title       : Link_Gstate
-- File        : global_reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 18 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure finds the next non-null pointer to link a new global state node too.
--               If all transitions are used, a message is output to the screen.  In this case another
--               model may be more appropriate, or the current procedure can be modified to account
--               for more links.
--=============================================================================================================
procedure Link_Gstate(trans          : in out scm_transition_type;
                      current_Gstate : in out Glink_type;
                      next_Gstate    : in out Glink_type;
                      new_one        : in out boolean) is
begin
   if (current_Gstate.link1.Glink = null) then
      current_Gstate.link1.Gtransition := trans;
      current_Gstate.link1.Glink := next_Gstate;
      current_Gstate.link1.new_node := new_one;
   else
      if (current_Gstate.link2.Glink = null) then
         current_Gstate.link2.Gtransition := trans;
         current_Gstate.link2.Glink := next_Gstate;
         current_Gstate.link2.new_node := new_one;
      else
         if (current_Gstate.link3.Glink = null) then
            current_Gstate.link3.Gtransition := trans;
            current_Gstate.link3.Glink := next_Gstate;
            current_Gstate.link3.new_node := new_one;
         else
            if (current_Gstate.link4.Glink = null) then
               current_Gstate.link4.Gtransition := trans;
               current_Gstate.link4.Glink := next_Gstate;
               current_Gstate.link4.new_node := new_one;
            else
               put_line("All transition pointers used!!!");
               scroll_pause;
            end if;
         end if;
      end if;
   end if;

end Link_Gstate;
```

```
   --------------------------------------------------------------------------
   --This procedure simply links the newly created global state node to the previous
   --node in the graph.
   --------------------------------------------------------------------------
   procedure Insert_Gstate(temp          : in out Gstate_record_type;
                           trans         : in out scm_transition_type;
                           current_Gstate : in out Glink_type;
                           counter       : in out natural) is
      temp_node  : Glink_type := new global_state_type;
      temp_ptr   : Glink_type;
      is_new     : boolean;
      Error_flag : boolean;
   begin
      temp_node.system_state_number := counter;
      temp_node.Gtuple := temp;
      is_new := true;
      Link_Gstate(trans, current_Gstate, temp_node, is_new);
      current_Gstate := temp_node;
   end Insert_Gstate;

   --------------------------------------------------------------------------
   --This procedure initializes the top global state.
   --------------------------------------------------------------------------

   procedure initialize_top_global_state(ptr : in out Glink_type) is
      temp : Gstate_record_type;
   begin
      temp.machine1_state.state_number := start_state_1;
      temp.machine2_state.state_number := start_state_2;
      ptr.Gtuple := temp;
   end initialize_top_global_state;

   --------------------------------------------------------------------------
begin
   create(reach, mode => out_file, name => "file.dat");
   temp_tuple_pointer := new global_state_type;
   temp_tuple_pointer.system_state_number := Gstate_counter;
   initialize_top_global_state(temp_tuple_pointer);
   top_Gstate := temp_tuple_pointer;
   parent_Gstate := top_Gstate;
   current_Gstate := parent_Gstate;
   M1_state := start_state_1;
   M2_state := start_state_2;
   output_Gtuple_to_file(top_Gstate.Gtuple, Gstate_counter);
   new_line;
   ------------------------------------------------
   begin
      main_loop :
         loop
         --find all valid transitions in current row of machine array 1
            position_holder1 := machine_array(1) (M1_state);
            row_holder1 := position_holder1;
            current_Gstate := parent_Gstate;
            --analyze all variables and determine which transitions to put on stack
```

```
transition_stack := Analyze_Predicates_Machine1(current_Gstate.Gtuple.machine1_state,
 current_Gstate.Gtuple.GLOBAL_VARIABLES);
while (not IsEmpty(transition_stack)) loop
--do the loop until the global transition stack is empty
   current_Gstate := parent_Gstate;
   position_holder1 := row_holder1;
   while (position_holder1 /= null) loop
   --analyze the current row of machine array until it reaches the null pointer
      if (position_holder1.transition = Top(transition_stack)) then
      --when a value is found then determine if it is a new node or not
      --if a node exists make a link in graph
      --if the node does not exist make a new node and add to graph
         toggle := true;
         position_holder1.visited := yes;
         Pop(transition_stack, current_transition);
         temp_Gstate.machine1_state := current_Gstate.Gtuple.machine1_state;
         temp_Gstate.machine2_state := current_Gstate.Gtuple.machine2_state;
         temp_Gstate.GLOBAL_VARIABLES := current_Gstate.Gtuple.GLOBAL_VARIABLES;
         temp_Gstate.machine1_state.state_number := position_holder1.next_Mstate;
         temp_Gstate.machine2_state.state_number := M2_state;
         Action(current_Gstate.Gtuple, current_transition, temp_Gstate);
         find_Gstate := search_for_Gtuple(top_Gstate, temp_Gstate);
         if (find_Gstate = null) then
            Gstate_counter := Gstate_counter + 1;
            Insert_Gstate(temp_Gstate, current_transition, current_Gstate, Gstate_counter);
            output_Gtuple_to_file(current_Gstate.Gtuple, Gstate_counter);
            Enqueue(Gpointer_queue, current_Gstate);
         else
            is_new := false;
            Link_Gstate(current_transition, current_Gstate, find_Gstate, is_new);
         end if;
         exit;
      else
         position_holder1 := position_holder1.Slink;
      end if;
   end loop;
   if (not IsEmpty(transition_stack) and (not toggle)) then
      Pop(transition_stack, current_transition);
   else
      toggle := false;
   end if;
end loop;
--find all valid transitions in current row of machine array 2
position_holder2 := machine_array(2) (M2_state);
row_holder2 := position_holder2;
current_Gstate := parent_Gstate;
--analyze all variables and determine which transitions to put on stack
transition_stack := Analyze_Predicates_Machine2(current_Gstate.Gtuple.machine2_state,
 current_Gstate.Gtuple.GLOBAL_VARIABLES);
```

```
         while (not IsEmpty(transition_stack)) loop
            position_holder2 := row_holder2;
            current_Gstate := parent_Gstate;
            --analyze the current row of machine array until it reaches the null pointer
            while (position_holder2 /= null) loop
               if (position_holder2.transition = Top(transition_stack)) then
                  --when a value is found then determine if it is a new node or not
                  --if a node exists make a link in graph
                  --if the node does not exist make a new node and add to graph
                  toggle := true;
                  position_holder2.visited := yes;
                  Pop(transition_stack, current_transition);
                  temp_Gstate.machine1_state := current_Gstate.Gtuple.machine1_state;
                  temp_Gstate.machine2_state := current_Gstate.Gtuple.machine2_state;
                  temp_Gstate.GLOBAL_VARIABLES := current_Gstate.Gtuple.GLOBAL_VARIABLES;
                  temp_Gstate.machine1_state.state_number := M1_state;
                  temp_Gstate.machine2_state.state_number := position_holder2.next_Mstate;
                  Action(current_Gstate.Gtuple, current_transition, temp_Gstate);
                  find_Gstate := search_for_Gtuple(top_Gstate, temp_Gstate);
                  if (find_Gstate = null) then
                     Gstate_counter := Gstate_counter + 1;
                     Insert_Gstate(temp_Gstate, current_transition, current_Gstate, Gstate_counter);
                     output_Gtuple_to_file(current_Gstate.Gtuple, Gstate_counter);
                     Enqueue(Gpointer_queue, current_Gstate);
                  else
                     is_new := false;
                     Link_Gstate(current_transition, current_Gstate, find_Gstate, is_new);
                  end if;
                  exit;
               else
                  position_holder2 := position_holder2.Slink;
               end if;
            end loop;
            if (not IsEmpty(transition_stack) and (not toggle)) then
               Pop(transition_stack, current_transition);
            else
               toggle := false;
            end if;
         end loop;
         if (IsEmpty(Gpointer_queue)) then
            exit;
         else
            parent_Gstate := First(Gpointer_queue);
            Dequeue(Gpointer_queue);
            M1_state := parent_Gstate.Gtuple.machine1_state.state_number;
            M2_state := parent_Gstate.Gtuple.machine2_state.state_number;
         end if;
      end loop main_loop;
      --when the pointer queue is full an exception is raised to exit the
      --main loop
exception
   when Gpointer_queue_package.QueueFull =>
      put("     The pointer queue is full at state: ");
      put(current_Gstate.system_state_number, width => 1);
```

```
            new_line;
            scroll_pause;
            Gpointer_queue_package.MakeEmpty(Gpointer_queue);
         when Gpointer_stack_package.StackFull =>
            put("     The pointer stack is full at state: ");
            put(current_Gstate.system_state_number, width => 1);
            new_line;
            Gpointer_stack_package.MakeEmpty(Gpointer_stack);
      end;
      Gstate_counter := Gstate_counter + 1;
      number_of_nodes := Gstate_counter;
      put("      Number of nodes in the Global Reachability  Graph: ");
      put(Gstate_counter, width => 1);
      new_line;
      close(reach);
   end build_Gstate_graph;
```

```
separate(main)
--=============================================================================================
-- Title       : clear_pointers
-- File        : global_reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 18 March 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Clear all pointers to apply to another user input file.
--=============================================================================================

procedure clear_pointers is
begin
   initial_system_state := null;
   initial_global_state := null;
   for i in 1 .. 2 loop
      for j in 0 .. 10 loop
         machine_array(i) (j) := null;
      end loop;
   end loop;
end clear_pointers;
```

```
separate(main)
--=============================================================================================================-
-- Title       : search_for_Gtuple
-- File        : global_search.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System .    : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : A function that searchs the reachability graph for a duplicated tuple.  Returns a pointer
--               value of the found tuple location, else returns null.
--=============================================================================================================-


function search_for_Gtuple(top : in Glink_type; tuple : in Gstate_record_type) return Glink_type is
   temp : Glink_type := null;
begin
   if (top = null) then
      return null;
   elsif (tuple=top.Gtuple) then
      return top;
   else
      if ((temp = null) and then (top.link1.new_node = true)) then
         temp := search_for_Gtuple(top.link1.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link2.new_node = true)) then
         temp := search_for_Gtuple(top.link2.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link3.new_node = true)) then
         temp := search_for_Gtuple(top.link3.Glink, tuple);
      end if;
      if ((temp = null) and then (top.link4.new_node = true)) then
         temp := search_for_Gtuple(top.link4.Glink, tuple);
      end if;
      return temp;
   end if;
end search_for_Gtuple;
```

```
separate(main)
--=============================================================================================
-- Title       : clear_pause
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain,  U. S. Army
-- Date        : 10 Feb 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Procedure called when default output screen reaches max.  Default max is 25 lines.
--=============================================================================================

procedure clear_pause is
   keyboard : string(1 .. 1);
   len      : natural;
begin
   new_line;
   get_line(keyboard, len);
   put(ASCII.ESC);
   put_line("[2J");
end clear_pause;
```

```
separate(main)
--=============================================================================================
-- Title       : scroll_pause
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 Feb 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Procedure called when default output screen reaches max.  Default max is 25 lines.
--               User is not prompted for ENTER.
--=============================================================================================
procedure scroll_pause is
   keyboard : string(1 .. 1);
   len      : natural;
begin
   new_line;
   get_line(keyboard, len);
end scroll_pause;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_machine_arrays
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 Feb 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Output the contents of each machine array to an output text file.  Accepts a pointer
--               to a three dimensional machine array (pointers to machine arrays).
--=============================================================================================================

procedure output_machine_arrays(machine_array : in out system_array_type) is
   place_holder : Slink_type;
   line_count   : integer := 7;
begin
   new_line(2);
   set_col(20);
   put("-----------------------------------------");
   set_col(20);
   put("|      Machine 1 Array Contents          |");
   set_col(20);
   put("-----------------------------------------");
   set_col(20);
   put("| From  |  To  |  Transition  | Executed |");
   set_col(20);
   put("-----------------------------------------");

   for curr_state1 in 0 .. 10 loop
      place_holder := machine_array(1) (curr_state1);
      while (place_holder /= null) loop
         line_count := line_count + 1;
         if ((line_count mod 20) = 0) then
            clear_pause;
         end if;
         set_col(20);
         put("|  " & integer'image(curr_state1));
         put("   |  " & integer'image(place_holder.next_Mstate) & "  |  ");
         put(place_holder.transition, set => lower_case);
         set_col(50);
         put("|   ");
         put(place_holder.visited, set => lower_case, width => 3);
         set_col(61);
         put("|");
         new_line;
         place_holder := place_holder.Slink;
      end loop;
   end loop;
```

```
      set_col(20);
      put("-----------------------------------------");
      new_line(2);
      set_col(20);
      put("-----------------------------------------");
      set_col(20);
      put("|       Machine 2 Array Contents          |");
      set_col(20);
      put("-----------------------------------------");
      set_col(20);
      put("| From  |  To  |  Transition  | Executed |");
      set_col(20);
      put("-----------------------------------------");
      for curr_state2 in 0 .. 10 loop
         place_holder := machine_array(2) (curr_state2);
         while (place_holder /= null) loop
            line_count := line_count + 1;
            if ((line_count mod 20) = 0) then
               clear_pause;
            end if;
            set_col(20);
            put("|   " & integer'image(curr_state2));
            put("    |   " & integer'image(place_holder.next_Mstate) & "   |   ");
            put(place_holder.transition, set => lower_case);
            set_col(50);
            put("|    ");
            put(place_holder.visited, set => lower_case, width => 3);
            set_col(61);
            put("|");
            new_line;
            place_holder := place_holder.Slink;
         end loop;
      end loop;

      set_col(20);
      put("-----------------------------------------");
      new_line;
   end output_machine_arrays;
```

```
separate(main)
--=============================================================================================
-- Title       : output_Gstate_transition
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 17 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs global state transition to default output.  Accepts a pointer to the global
--               state graph as input. The output contains transition type and message.
--=============================================================================================
procedure output_Gstate_transition(Gstate_pointer : in out Glink_type) is
begin
   if (Gstate_pointer.link1.Glink /= null) then
      set_col(Gcolumn_set + 68);
      put(Gstate_pointer.link1.Gtransition, width => transit_len, set => lower_case);
      put(Gstate_pointer.link1.Glink.system_state_number, width => 4);
      new_line;
   end if;
   if (Gstate_pointer.link2.Glink /= null) then
      set_col(Gcolumn_set + 68);
      put(Gstate_pointer.link2.Gtransition, width => transit_len, set => lower_case);
      put(Gstate_pointer.link2.Glink.system_state_number, width => 4);
      new_line;
   end if;
   if (Gstate_pointer.link3.Glink /= null) then
      set_col(Gcolumn_set + 68);
      put(Gstate_pointer.link3.Gtransition, width => transit_len, set => lower_case);
      put(Gstate_pointer.link3.Glink.system_state_number, width => 4);
      new_line;
   end if;
   if (Gstate_pointer.link4.Glink /= null) then
      set_col(Gcolumn_set + 68);
      put(Gstate_pointer.link4.Gtransition, width => transit_len, set => lower_case);
      put(Gstate_pointer.link4.Glink.system_state_number, width => 4);
      new_line;
   end if;
end output_Gstate_transition;
```

```
separate(main)
--=============================================================================================
-- Title       : output_Gstates
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 10 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs the entire reachability graph.  Calls procedures output_Gstates and output_
--               Gstate_node.  Accepts pointer to top global state and number of nodes in the graph
--               as variables. Output is directed to default ouput (CRT).
--=============================================================================================

procedure output_Gstates(top_Gstate : in out Glink_type) is
   Error_flag : boolean    := true;
   current    : Glink_type := top_Gstate;
begin
   if (top_Gstate = null) then
      return;
   else
      output_Gstate_node(top_Gstate, Error_flag);
      if (not Error_flag) then
         output_Gstate_transition(top_Gstate);
      end if;
      if (top_Gstate.link1.new_node = true) then
         output_Gstates(top_Gstate.link1.Glink);
      end if;
      if (top_Gstate.link2.new_node = true) then
         output_Gstates(top_Gstate.link2.Glink);
      end if;
      if (top_Gstate.link3.new_node = true) then
         output_Gstates(top_Gstate.link3.Glink);
      end if;
      if (top_Gstate.link4.new_node = true) then
         output_Gstates(top_Gstate.link4.Glink);
      end if;
   end if;
end output_Gstates;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_analysis
-- File        : global_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 17 March 19 92
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure builds the machine arrays, and produces the global reachability graph.
--               Parameters are described in the header of each procedure.
--=============================================================================================================

procedure output_analysis is
begin
   output_machine_arrays(machine_array);
   clear_pause;
   build_Gstate_graph(machine_array, start_state_1, start_state_2, initial_global_state);
   clear_pause;
   new_line(2);
   put_line("                                    Global State GRAPH    ");
   new_line;
   output_line_count := 1;
   output_Gstates(initial_global_state);
   clear_pause;
   -- new_line(2);
   -- put_line("              WE ARE OUTTA HERE!!!!!!!!!");
   new_line(2);
   put_line("                     System State GRAPH    ");
   new_line;
   output_line_count := 1;
   --test;

   build_Sstate_graph(initial_global_state, initial_system_state);
   output_Sstates(initial_system_state);
   new_line;
   clear_pause;
   output_machine_arrays(machine_array);
   scroll_pause;
   --   new_line(2);
   --   new_line(reach, 2);
   --   put_line(reach, "                          Reachability Graph");
   --   output_Gstates_to_file(initial_global_state, number_of_nodes);
   --   new_line(reach);
   --  output_machine_arrays_to_file(machine_array);
end output_analysis;
```

```
separate(main)
-------------------------------------------------------------------------------------------------------------
-- Title       : build_Sstate_graph
-- File        : system_reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : This procedure builds the system reachability graph from the global reachability graph.  Each
--               system state node can have up to 4 transitions.  The algorithm for this procedure is as follows.
--               Based on the initial state of the global reachability graph an initial state is made in the system
--               reachability graph.  Values are assigned to the system tuple and values are placed in the four
--               outgoing transitions.  Whenever a temporary system state tuple is filled the machine states are
--               added as well as the outgoing transitions.  The global state graph is traversed in a breadth first
--               manner.  A temporary system state tuple is made and a check is done in the system state graph is
--               made to determine if the state already exits.  If the state already exits a link is made from the
--               current system state to the found state.  If the temporary state value is not found the temporary
--               node is linked into the graph.  The appropriate link, system state, and global state are 'remembered'
--               with the use of queues to hold the values.   A pointer is retrieved from the queue after a scan is
--               done while traversing the global state graph. This procedure terminates whenever there are no values
--               in the global pointer/system state/link queues. In other words, this procedure ends when each node
--               in the global graph has been visited and each transition from each of the global nodes has been
--               analyzed.  As can be demonstrated, SCM system reachability graphs can grow without bound.
--               A max pointer queue size is introduced to eliminate unboundedness.
-------------------------------------------------------------------------------------------------------------

procedure build_Sstate_graph(top_Gstate : in out Glink_type; top_Sstate : in out Syslink_type) is
   parent_Sstate, current_Sstate, find_Sstate, temp_tuple_pointer : Syslink_type := null;
   current_Gstate                                                 : Glink_type   := null;
   Sstate_counter                                                 : natural      := 0;
   temp_Glink                                                     : Gstate_transition_type;
   temp_Sstate                                                    : Syslink_type;
   next_state                                                     : natural;    --returned value from a transition
   is_new                                                         : boolean;
   current_transition                                             : scm_transition_type;
   hold_Srecord                                                   : system_state_type;
   loop_count                                                     : integer      := 0;
   linkno                                                         : integer      := 1;
```

```
   --=============================================================================================================-
   -- Title       : Link_Sstate
   -- File        : system_reachability.a
   -- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
   -- Date        : 20 March 1992
   -- Revised     :
   -- System      : Sun 4 Workstation
   -- Compiler    : Verdix ADA, Version 6.0
   -- Description : This procedure finds the next non-null pointer to link a new global state node too.
   --               If all transitions are used, a message is output to the screen.  In this case another
   --               model may be more appropriate, or the current procedure can be modified to account
   --               for more links.
   --=============================================================================================================-

   procedure Link_Sstate(current_Strans : in out Sstate_transition_type; next_Sstate : in out Syslink_type) is
   begin
      if (current_Strans.Syslink = null) then
         current_Strans.Syslink := next_Sstate;
         current_Strans.new_node := false;
      end if;
   end Link_Sstate;

   ---------------------------------------------------------------------------
   --This procedure simply links the newly created system state node to the previous
   --node in the graph.
   ---------------------------------------------------------------------------
   procedure Insert_Sstate(Shold         : in out system_state_type;
                           current_Strans : in out Sstate_transition_type;
                           counter        : in out integer) is
      temp_node : Syslink_type := new system_state_type;
      is_new    : boolean;
   begin
      temp_node.system_state_number := counter;
      temp_node.Stuple := Shold.Stuple;
      temp_node.link1 := Shold.link1;
      temp_node.link2 := Shold.link2;
      temp_node.link3 := Shold.link3;
      temp_node.link4 := Shold.link4;
      current_Strans.Syslink := temp_node;
      current_Strans.new_node := true;
   end Insert_Sstate;


   ---------------------------------------------------------------------------
begin
   --build the initial system state
   current_Sstate := new system_state_type;
   current_Gstate := top_Gstate;
   current_Sstate.system_state_number := Sstate_counter;
```

```
current_Sstate.Stuple.machine1_state := current_Gstate.Gtuple.machine1_state.state_number;
current_Sstate.Stuple.machine2_state := current_Gstate.Gtuple.machine2_state.state_number;
top_Sstate := current_Sstate;
--save the value of the used transitions from the global reachability graph
--for later analysis
if (current_Gstate.link1.Gtransition /= unused) then
   current_Gstate.link1.visited := true;
   top_Sstate.link1.Stransition := current_Gstate.link1.Gtransition;
   Enqueue(Sstate, current_Sstate);
   Enqueue(link_queue, 1);
   Enqueue(Gtrans_queue, current_Gstate.link1);
end if;
if (current_Gstate.link2.Gtransition /= unused) then
   current_Gstate.link2.visited := true;
   top_Sstate.link2.Stransition := current_Gstate.link2.Gtransition;
   Enqueue(Sstate, current_Sstate);
   Enqueue(link_queue, 2);
   Enqueue(Gtrans_queue, current_Gstate.link2);
end if;
if (current_Gstate.link3.Gtransition /= unused) then
   top_Sstate.link3.Stransition := current_Gstate.link3.Gtransition;
   current_Gstate.link3.visited := true;
   Enqueue(Sstate, current_Sstate);
   Enqueue(link_queue, 3);
   Enqueue(Gtrans_queue, current_Gstate.link3);
end if;
if (current_Gstate.link4.Gtransition /= unused) then
   current_Gstate.link4.visited := true;
   top_Sstate.link4.Stransition := current_Gstate.link4.Gtransition;
   Enqueue(Sstate, current_Sstate);
   Enqueue(link_queue, 4);
   Enqueue(Gtrans_queue, current_Gstate.link4);
end if;

begin
   loop
      --begin analysis of states held in the event queues
      loop_count := loop_count + 1;
      temp_Glink := First(Gtrans_queue);
      current_Sstate := First(Sstate);
      linkno := First(link_queue);
      current_transition := temp_Glink.Gtransition;
      current_Gstate := temp_Glink.Glink;
      --load the temporary record to be used in the search for a matching tuple
      hold_Srecord.Stuple.machine1_state := current_Gstate.Gtuple.machine1_state.state_number;
      hold_Srecord.Stuple.machine2_state := current_Gstate.Gtuple.machine2_state.state_number;
      hold_Srecord.link1.Stransition := current_Gstate.link1.Gtransition;
      hold_Srecord.link1.new_node := true;
      hold_Srecord.link2.Stransition := current_Gstate.link2.Gtransition;
      hold_Srecord.link2.new_node := true;
      hold_Srecord.link3.Stransition := current_Gstate.link3.Gtransition;
      hold_Srecord.link3.new_node := true;
      hold_Srecord.link4.Stransition := current_Gstate.link4.Gtransition;
      hold_Srecord.link4.new_node := true;
```

```
--search for the tuple in the system reachability graph
find_Sstate := search_for_Stuple(top_Sstate, hold_Srecord);
if (find_Sstate = null) then
   Sstate_counter := Sstate_counter + 1;
   --look at the correct link
   case linkno is
      when 1 =>
         Insert_Sstate(hold_Srecord, current_Sstate.link1, Sstate_counter);
         current_Sstate := current_Sstate.link1.Syslink;
      when 2 =>
         Insert_Sstate(hold_Srecord, current_Sstate.link2, Sstate_counter);
         current_Sstate := current_Sstate.link2.Syslink;
      when 3 =>
         Insert_Sstate(hold_Srecord, current_Sstate.link3, Sstate_counter);
         current_Sstate := current_Sstate.link3.Syslink;
      when 4 =>
         Insert_Sstate(hold_Srecord, current_Sstate.link4, Sstate_counter);
         current_Sstate := current_Sstate.link4.Syslink;
      when others =>
         put_line("whats up doc?");
         scroll_pause;
   end case;

   --enqueue global state transitions that have not been analyzed
   if ((current_Gstate.link1.Glink /= null) and (current_Gstate.link1.visited = false)) then
      current_Gstate.link1.visited := true;
      Enqueue(Gtrans_queue, current_Gstate.link1);
      Enqueue(link_queue, 1);
      Enqueue(Sstate, current_Sstate);
   end if;
   if ((current_Gstate.link2.Glink /= null) and (current_Gstate.link2.visited = false)) then
      current_Gstate.link2.visited := true;
      Enqueue(Gtrans_queue, current_Gstate.link2);
      Enqueue(link_queue, 2);
      Enqueue(Sstate, current_Sstate);
   end if;
   if ((current_Gstate.link3.Glink /= null) and (current_Gstate.link3.visited = false)) then
      current_Gstate.link3.visited := true;
      Enqueue(Gtrans_queue, current_Gstate.link3);
      Enqueue(link_queue, 3);
      Enqueue(Sstate, current_Sstate);
   end if;
   if ((current_Gstate.link4.Glink /= null) and (current_Gstate.link4.visited = false)) then
      current_Gstate.link4.visited := true;
      Enqueue(Gtrans_queue, current_Gstate.link4);
      Enqueue(link_queue, 4);
      Enqueue(Sstate, current_Sstate);
   end if;
else
```

```
            --link the appropriate transition link
            case linkno is
               when 1 =>
                  Link_Sstate(current_Sstate.link1, find_Sstate);
                  current_Sstate := current_Sstate.link1.Syslink;
               when 2 =>
                  Link_Sstate(current_Sstate.link2, find_Sstate);
                  current_Sstate := current_Sstate.link2.Syslink;
               when 3 =>
                  Link_Sstate(current_Sstate.link3, find_Sstate);
                  current_Sstate := current_Sstate.link3.Syslink;
               when 4 =>
                  Link_Sstate(current_Sstate.link4, find_Sstate);
                  current_Sstate := current_Sstate.link4.Syslink;
               when others =>
                  put_line("whats up doc?");
                  scroll_pause;
            end case;
            --enqueue global transitions that have not been analyzed
            if ((current_Gstate.link1.Glink /= null) and (current_Gstate.link1.visited = false)) then
               current_Gstate.link1.visited := true;
               Enqueue(Gtrans_queue, current_Gstate.link1);
               Enqueue(link_queue, 1);
               Enqueue(Sstate, current_Sstate);
            end if;
            if ((current_Gstate.link2.Glink /= null) and (current_Gstate.link2.visited = false)) then
               current_Gstate.link2.visited := true;
               Enqueue(Gtrans_queue, current_Gstate.link2);
               Enqueue(link_queue, 2);
               Enqueue(Sstate, current_Sstate);
            end if;
            if ((current_Gstate.link3.Glink /= null) and (current_Gstate.link3.visited = false)) then
               current_Gstate.link3.visited := true;
               Enqueue(Gtrans_queue, current_Gstate.link3);
               Enqueue(link_queue, 3);
               Enqueue(Sstate, current_Sstate);
            end if;
            if ((current_Gstate.link4.Glink /= null) and (current_Gstate.link4.visited = false)) then
               current_Gstate.link4.visited := true;
               Enqueue(Gtrans_queue, current_Gstate.link4);
               Enqueue(link_queue, 4);
               Enqueue(Sstate, current_Sstate);
            end if;
         end if;
         --delete the top state from each queue, at this point they have been analyzed
         Dequeue(Gtrans_queue);
         Dequeue(link_queue);
         Dequeue(Sstate);
      end loop;

   --when the queues are empty the main loop will raise the following exception handler
   exception
      when Gtrans_stack_package.StackEmpty =>
         put_line("hi ho hi ho");
```

```
         scroll_pause;
         null;
      when Gtrans_queue_package.QueueEmpty =>
         null;
      when others =>
         put_line("error in the system reachability procedure");
         scroll_pause;
   end;
   put("       Number of nodes in the System State Graph: ");
   Sstate_counter := Sstate_counter + 1;
   put(Sstate_counter);
   new_line;
   scroll_pause;
end build_Sstate_graph;

separate(main)

-------------------------------------------------------------------------------------------------
-- Title       : IsSysStateEqual
-- File        : system_reachability.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Compares two system state tuples for equality.  Returns a boolean(true/false).
--               Equality of transitions is determined by the combined alphanumeric (ASCII)
--               value of all four transitions.
-------------------------------------------------------------------------------------------------

function IsSysStateEqual(top : in Syslink_type; current : in system_state_type) return boolean is

   top_temp     : integer := 0;
   current_temp : integer := 0;
   str_len      : integer;
   str          : string(1 .. 10);

   function enum_value(InString : in string) return integer is
      tot : integer := 0;
   begin
      for i in InString'range loop
         tot := tot + (character'pos(InString(i)));
      end loop;
      return tot;
   end enum_value;


begin
   if (top.Stuple = current.Stuple) then
```

```
      trans_enum_io.put(str, top.link1.Stransition, set => lower_case);
      top_temp := top_temp + enum_value(str);
      trans_enum_io.put(str, top.link2.Stransition, set => lower_case);
      top_temp := top_temp + enum_value(str);
      trans_enum_io.put(str, top.link3.Stransition, set => lower_case);
      top_temp := top_temp + enum_value(str);
      trans_enum_io.put(str, top.link4.Stransition, set => lower_case);
      top_temp := top_temp + enum_value(str);
      trans_enum_io.put(str, current.link1.Stransition, set => lower_case);
      current_temp := current_temp + enum_value(str);
      trans_enum_io.put(str, current.link2.Stransition, set => lower_case);
      current_temp := current_temp + enum_value(str);
      trans_enum_io.put(str, current.link3.Stransition, set => lower_case);
      current_temp := current_temp + enum_value(str);
      trans_enum_io.put(str, current.link4.Stransition, set => lower_case);
      current_temp := current_temp + enum_value(str);
      if (top_temp = current_temp) then
         return true;
      else
         return false;
      end if;
   else
      return false;
   end if;
end IsSysStateEqual;
```

```
separate(main)
--=============================================================================================
-- Title       : search_for_Stuple
-- File        : system_search.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : A function that searchs the reachability graph for a duplicated tuple.  Returns a pointer
--               value of the found tuple location, else returns null.
--=============================================================================================

function search_for_Stuple(top : in Syslink_type; current : in system_state_type) return Syslink_type is
   temp : Syslink_type := null;
begin
   if (top = null) then
      return null;
   else
      if (IsSysStateEqual(top, current)) then
         return top;
      else
         if ((temp = null) and then (top.link1.new_node = true)) then
            temp := search_for_Stuple(top.link1.Syslink, current);
         end if;
         if ((temp = null) and then (top.link2.new_node = true)) then
            temp := search_for_Stuple(top.link2.Syslink, current);
         end if;
         if ((temp = null) and then (top.link3.new_node = true)) then
            temp := search_for_Stuple(top.link3.Syslink, current);
         end if;
         if ((temp = null) and then (top.link4.new_node = true)) then
            temp := search_for_Stuple(top.link4.Syslink, current);
         end if;
         return temp;
      end if;
   end if;
end search_for_Stuple;
```

```
separate(main)
--=============================================================================================
-- Title       : output_Stuple
-- File        : system_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs global state transition to default output.  Accepts a pointer to the system
--               state graph as input. The output contains transition type and the machine states.
--=============================================================================================

procedure output_Stuple(tuple : in out Sstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state) & " ,");
   put(integer'image(tuple.machine2_state) & " ]  ");
end output_Stuple;

separate(main)
--=============================================================================================
-- Title       : output_Sstate_node
-- File        : system_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs system node and formats for display.  If there is no transition out of the
--               state, a deadlock message is displayed.
--=============================================================================================

procedure output_Sstate_node(Sstate_pointer : in out Syslink_type; Error_flag : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   Scolumn_set := 10;
   set_col(Scolumn_set);
   put(Sstate_pointer.system_state_number, width => 3);
   output_Stuple(Sstate_pointer.Stuple);
   if ((Sstate_pointer.link1.Syslink = null) and then (Sstate_pointer.link2.Syslink = null) and then
    (Sstate_pointer.link3.Syslink = null) and then (Sstate_pointer.link4.Syslink = null)) then
      Error_flag := true;
      set_col(Scolumn_set + 19);
      put("****** DEADLOCK STATE ***");
   else
      Error_flag := false;
   end if;
end output_Sstate_node;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_Sstate_transition
-- File        : system_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs system state transition to default output.  Accepts a pointer to the system
--               state graph as input. The output contains transition type and the next tuple on the
--               link.
--=============================================================================================================

procedure output_Sstate_transition(Sstate_pointer : in out Syslink_type) is
begin
   if (Sstate_pointer.link1.Stransition /= unused) then
      set_col(Scolumn_set + 19);    --this was 19
      put(Sstate_pointer.link1.Stransition, width => transit_len, set => lower_case);
      if (Sstate_pointer.link1.Syslink /= null) then
         output_Stuple(Sstate_pointer.link1.Syslink.Stuple);
         put(Sstate_pointer.link1.Syslink.system_state_number, width => 1);
      end if;
      new_line;
   end if;
   if (Sstate_pointer.link2.Stransition /= unused) then
      set_col(Scolumn_set + 19);
      put(Sstate_pointer.link2.Stransition, width => transit_len, set => lower_case);
      if (Sstate_pointer.link2.Syslink /= null) then
         output_Stuple(Sstate_pointer.link2.Syslink.Stuple);
         put(Sstate_pointer.link2.Syslink.system_state_number, width => 1);
      end if;
      new_line;
   end if;
   if (Sstate_pointer.link3.Stransition /= unused) then
      set_col(Scolumn_set + 19);
      put(Sstate_pointer.link3.Stransition, width => transit_len, set => lower_case);
      if (Sstate_pointer.link3.Syslink /= null) then
         output_Stuple(Sstate_pointer.link3.Syslink.Stuple);
         put(Sstate_pointer.link3.Syslink.system_state_number, width => 1);
      end if;
      new_line;
   end if;
   if (Sstate_pointer.link4.Stransition /= unused) then
      set_col(Scolumn_set + 19);
      put(Sstate_pointer.link4.Stransition, width => transit_len, set => lower_case);
      if (Sstate_pointer.link4.Syslink /= null) then
         output_Stuple(Sstate_pointer.link4.Syslink.Stuple);
         put(Sstate_pointer.link4.Syslink.system_state_number, width => 1);
```

```
      end if;
      new_line;
   end if;
end output_Sstate_transition;
```

```
separate(main)
--=============================================================================================================
-- Title       : output_Sstates
-- File        : system_output.a
-- Author      : Matthew J. Rothlisberger, Captain, U. S. Army
-- Date        : 20 March 1992
-- Revised     :
-- System      : Sun 4 Workstation
-- Compiler    : Verdix ADA, Version 6.0
-- Description : Outputs the entire reachability graph.  Calls procedures output_Sstates, output_
--               Sstate_node, and output_Stuple.  Accepts pointer to top system state. Output is
--               directed to default ouput (CRT).
--=============================================================================================================

procedure output_Sstates(top_Sstate : in out Syslink_type) is
   Error_flag : boolean      := true;
   current    : Syslink_type := top_Sstate;
begin
   if (top_Sstate = null) then
      return;
   else
      output_Sstate_node(top_Sstate, Error_flag);
      if (not Error_flag) then
         output_Sstate_transition(top_Sstate);
      end if;
      if ((top_Sstate.link1.Syslink /= null) and then (top_Sstate.link1.new_node = true)) then
         output_Sstates(top_Sstate.link1.Syslink);
      end if;
      if ((top_Sstate.link2.Syslink /= null) and then (top_Sstate.link2.new_node = true)) then
         output_Sstates(top_Sstate.link2.Syslink);
      end if;
      if ((top_Sstate.link3.Syslink /= null) and then (top_Sstate.link3.new_node = true)) then
         output_Sstates(top_Sstate.link3.Syslink);
      end if;
      if ((top_Sstate.link4.Syslink /= null) and then (top_Sstate.link4.new_node = true)) then
         output_Sstates(top_Sstate.link4.Syslink);
      end if;
   end if;
end output_Sstates;
```

APPENDIX C (CFSM) ALTERNATING BIT

INPUT (FSM)

```
start
machine 1 -------------MACHINE 1----------
state 1
trans -X 2
state 2
trans +A 3
state 3
trans -Y 4
state 4
trans +B 1
machine 2 -------------MACHINE 2-------------
state 1
trans +X 2
state 2
trans -A 3
state 3
trans +Y 4
state 4
trans -B 1
initial_state 1 1
finish
```

OUTPUT

REACHABILITY ANALYSIS of : output.alt

Reachability Graph

1	[1 , E , E , 1]	-X	[2 , X , E , 1] 2
2	[2 , X , E , 1]	+X	[2 , E , E , 2] 3
3	[2 , E , E , 2]	-A	[2 , E , A , 3] 4
4	[2 , E , A , 3]	+A	[3 , E , E , 3] 5
5	[3 , E , E , 3]	-Y	[4 , Y , E , 3] 6
6	[4 , Y , E , 3]	+Y	[4 , E , E , 4] 7
7	[4 , E , E , 4]	-B	[4 , E , B , 1] 8
8	[4 , E , B , 1]	+B	[1 , E , E , 1] 1

Machine 1 Array Contents

From	To	Transition	Executed
1	2	snd X	yes
2	3	rcv A	yes
3	4	snd Y	yes
4	1	rcv B	yes

Machine 2 Array Contents

From	To	Transition	Executed
1	2	rcv X	yes
2	3	snd A	yes
3	4	rcv Y	yes
4	1	snd B	yes

* The nodes generated by the analysis were done in a breadth first manner

APPENDIX D (CFSM) SLIDING WINDOW

INPUT(FSM)

```
start
machine 1
state 1
trans -X 2
state 2
trans +B 7
trans -Y 3
state 3
trans +B 8
trans +C 4
state 4
trans -Z 5
state 5
trans -X 6
trans +A 1
state 6
trans +A 2
trans +B 7
state 7
trans -Y 8
state 8
trans -Z 9
state 9
trans +C 5
trans +A 1
machine 2
state 1
trans +X 2
state 2
trans +Y 3
state 3
trans -C 4
state 4
trans +Z 5
state 5
trans +X 6
state 6
trans -B 7
state 7
trans +Y 8
state 8
trans +Z 9
state 9
trans -A 1
initial_state 1 1
finish
```

OUTPUT

REACHABILITY ANALYSIS of : sliding_window

Reachability Graph

1	[1 , E , E , 1]	-X	[2 , X , E , 1]	2
2	[2 , X , E , 1]	-Y	[3 , XY , E , 1]	3
		+X	[2 , E , E , 2]	4
3	[3 , XY , E , 1]	+X	[3 , Y , E , 2]	5
5	[3 , Y , E , 2]	+Y	[3 , E , E , 3]	6
6	[3 , E , E , 3]	-C	[3 , E , C , 4]	7
7	[3 , E , C , 4]	+C	[4 , E , E , 4]	8
8	[4 , E , E , 4]	-Z	[5 , Z , E , 4]	9
9	[5 , Z , E , 4]	-X	[6 , ZX , E , 4]	10
		+Z	[5 , E , E , 5]	11
10	[6 , ZX , E , 4]	+Z	[6 , X , E , 5]	12
12	[6 , X , E , 5]	+X	[6 , E , E , 6]	13
13	[6 , E , E , 6]	-B	[6 , E , B , 7]	14
14	[6 , E , B , 7]	+B	[7 , E , E , 7]	15
15	[7 , E , E , 7]	-Y	[8 , Y , E , 7]	16
16	[8 , Y , E , 7]	-Z	[9 , YZ , E , 7]	17
		+Y	[8 , E , E , 8]	18
17	[9 , YZ , E , 7]	+Y	[9 , Z , E , 8]	19
19	[9 , Z , E , 8]	+Z	[9 , E , E , 9]	20
20	[9 , E , E , 9]	-A	[9 , E , A , 1]	21
21	[9 , E , A , 1]	+A	[1 , E , E , 1]	1
18	[8 , E , E , 8]	-Z	[9 , Z , E , 8]	19
11	[5 , E , E , 5]	-X	[6 , X , E , 5]	12
4	[2 , E , E , 2]	-Y	[3 , Y , E , 2]	5

Machine 1 Array Contents

From	To	Transition	Executed
1	2	snd X	yes
2	7	rcv B	no
2	3	snd Y	yes
3	8	rcv B	no
3	4	rcv C	yes
4	5	snd Z	yes
5	6	snd X	yes
5	1	rcv A	no
6	2	rcv A	no
6	7	rcv B	yes
7	8	snd Y	yes
8	9	snd Z	yes
9	5	rcv C	no
9	1	rcv A	yes

Machine 2 Array Contents

From	To	Transition	Executed
1	2	rcv X	yes
2	3	rcv Y	yes
3	4	snd C	yes
4	5	rcv Z	yes
5	6	rcv X	yes
6	7	snd B	yes
7	8	rcv Y	yes
8	9	rcv Z	yes
9	1	snd A	yes

* The nodes generated by the analysis were done in a breadth first manner

APPENDIX E (SCM) GO_BACK_N, W=1

INPUT (FSM)

```
start
machine 1
state 0
trans snd_data 1
state 1
trans rcv_ack0 0
machine 2
state 0
trans rcv_data 1
state 1
trans snd_ack 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is

   type scm_transition_type  is (snd_data1, rcv_data1, snd_ack1,
                                 rcv_ack1, adv_win1, unused);
   type buffer_type   is (d1,e,a1);
   type boolean_type    is (t,f);

   type buffer_array_type is array(1..1) of buffer_type;
   type boolean_array_type is array(1..1) of boolean_type;

   type machine1_state_type is
      record
         state_number      : natural             := 0;
         out_buffer        : buffer_array_type   := (others=>d1);
         ack_rec           : boolean_array_type  := (others=>f);
         current           : integer range 1..1  := 1;
   end record;

   type machine2_state_type is
      record
         state_number      : natural             := 0;
         in_buffer         : buffer_array_type    := (others=>e);
         pkt_rec           : boolean_array_type   := (others=>f);
         current           : integer range 1..1   := 1;
   end record;

   type global_variable_type is
      record
         DATA              : buffer_type         := e;
    end record;

end definitions;
```

PREDICATE-ACTION

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if (local.out_buffer(1) /= E) then
          Push(transition_stack,snd_data1);
      end if;
      if ((local.ack_rec(1)=f) and GLOBAL.DATA=A1) then
          Push(transition_stack,rcv_ack1);
      end if;
      Push(transition_stack,adv_win1);
      return transition_stack;
end Analyze_Predicates_Machine1;

separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA = D1) and (local.pkt_rec(1)=f)) then
          Push(transition_stack,rcv_data1);
      end if;
      if (local.pkt_rec(1)=t) then
         Push(transition_stack,snd_ack1);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2;

separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
          when (snd_data1) =>
             out_system_state.GLOBAL_VARIABLES.DATA :=
                 in_system_state.machine1_state.out_buffer(1);
          when (rcv_ack1) =>
             out_system_state.machine1_state.ack_rec(1) := t;
              out_system_state.GLOBAL_VARIABLES.DATA := e;
             out_system_state.machine1_state.current := 1;
          when (rcv_data1) =>
              out_system_state.machine2_state.in_buffer(1) :=
                 in_system_state.GLOBAL_VARIABLES.DATA;
              out_system_state.GLOBAL_VARIABLES.DATA := e;
              out_system_state.machine2_state.pkt_rec(1):= t;
          when (snd_ack1) =>
             out_system_state.GLOBAL_VARIABLES.DATA := a1;
             out_system_state.machine2_state.pkt_rec(1) := f;
              out_system_state.machine2_state.in_buffer(1) := e;
         when (adv_win1)   =>
              out_system_state.ma-
chine1_state.ack_rec(in_system_state.machine1_state.current) := f;
          when others =>
             put_line("There is an error in the Action procedure");
      end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number));
   put(" ," & integer'image(tuple.machine2_state.state_number));
   put(" , ");
   put(tuple.machine1_state.out_buffer(1), width => 1);
   put(" , ");
   put(tuple.machine1_state.ack_rec(1),width=>2);
   put(" , ");
   put(tuple.machine2_state.in_buffer(1), width => 1);
   put(" , ");
   put(tuple.machine2_state.pkt_rec(1),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA, width =>2);
   put(" ]");
end output_Gtuple;


separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number));
   put(reach," ," & integer'image(tuple.machine2_state.state_number));
   put(reach," , ");
   put(reach,tuple.machine1_state.out_buffer(1), width => 1);
   put(reach," , ");
   put(reach,tuple.machine1_state.ack_rec(1),width=>2);
   put(reach," , ");
   put(reach,tuple.machine2_state.in_buffer(1), width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.pkt_rec(1),width=>2);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA, width =>2);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;


separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
      Error_flag := true;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

OUTPUT

```
                REACHABILITY ANALYSIS of : go_back_n_w1

                                Global State GRAPH

0   [ 0 , 0 , 0 , 1 , 0 , 1 , E , -1 , -1 ]    snd_data    1
1   [ 1 , 0 , 1 , 1 , 0 , 1 , D ,  0 , -1 ]    rcv_data    2
2   [ 1 , 1 , 1 , 1 , 1 , 1 , E , -1 , -1 ]    snd_ack     3
3   [ 1 , 0 , 1 , 1 , 1 , 1 , E , -1 ,  1 ]    rcv_ack0    4
4   [ 0 , 0 , 1 , 1 , 1 , 1 , E , -1 , -1 ]    snd_data    5
5   [ 1 , 0 , 0 , 1 , 1 , 1 , D ,  1 , -1 ]    rcv_data    6
6   [ 1 , 1 , 0 , 1 , 0 , 1 , E , -1 , -1 ]    snd_ack     7
7   [ 1 , 0 , 0 , 1 , 0 , 1 , E , -1 ,  0 ]    rcv_ack0    0


                        System State GRAPH

 0   [ 0 , 0 ]    snd_data    [ 1 , 0 ]  1
 1   [ 1 , 0 ]    rcv_data    [ 1 , 1 ]  2
 2   [ 1 , 1 ]    snd_ack     [ 1 , 0 ]  3
 3   [ 1 , 0 ]    rcv_ack0    [ 0 , 0 ]  0


-----------------------------------------
|       Machine 1 Array Contents        |
-----------------------------------------
| From  |  To  |  Transition  | Executed |
-----------------------------------------
|   0   |   1  |  snd_data    |   yes    |
|   1   |   0  |  rcv_ack0    |   yes    |
-----------------------------------------

-----------------------------------------
|       Machine 2 Array Contents        |
-----------------------------------------
| From  |  To  |  Transition  | Executed |
-----------------------------------------
|   0   |   1  |  rcv_data    |   yes    |
|   1   |   0  |  snd_ack     |   yes    |
-----------------------------------------
```

APPENDIX F (SCM) GO_BACK_N, W=2

INPUT (FSM)

```
start
machine 1
state 0
trans snd_data 1
state 1
trans snd_data 2
trans rcv_ack0 0
state 2
trans rcv_ack0 0
trans rcv_ack1 1
machine 2
state 0
trans rcv_data 1
state 1
trans rcv_data 2
trans snd_ack 0
state 2
trans snd_ack 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is


   type scm_transition_type  is (snd_data, rcv_data,
                                 snd_ack,  rcv_ack0,
                                rcv_ack1, unused);

   type buffer_type   is (d0,d1,e,a);
   type buffer_array_type is array(1..2) of buffer_type;
   type seq_array_type is array(1..2) of integer range -1..2;

   type machine1_state_type is
      record
          state_number      : natural := 0;
          Sdata             : buffer_array_type := (d0,d1);
          seq               : integer range 0..2 := 0;
          i                 : integer range 1..2 := 1;
   end record;

   type machine2_state_type is
      record
          state_number       : natural := 0;
          Rdata             : buffer_type := e;
          exp               : integer range 0..2 := 0;
          j                 : integer range 1..2 := 1;
   end record;

   type global_variable_type is
      record
          DATA              : buffer_array_type   := (e,e);
          SEQ               : seq_array_type      := (-1,-1);
          ACK               : integer range -1..2 := -1;
    end record;

end definitions;
```

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
      temp1 : integer := GLOBAL.ACK + 0;
      temp2 : integer := (GLOBAL.ACK + 1)mod 3;
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.i) = E) and (GLOBAL.SEQ(local.i) = -1)) then
         Push(transition_stack,snd_data);
      end if;
      if ((temp1 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack0);
      end if;
      if ((temp2 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack1);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine1;


separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.j)/=E) and (GLOBAL.SEQ(local.j) = local.exp)) then
         Push(transition_stack,rcv_data);
      end if;
      if (GLOBAL.DATA(local.j)=E) then
        Push(transition_stack,snd_ack);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2;

separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
          when (snd_data) =>
            out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine1_state.i) :=
                in_system_state.machine1_state.Sdata(in_system_state.machine1_state.i);
             out_system_state.GLOBAL_VARIABLES.SEQ(in_system_state.machine1_state.i) :=
                in_system_state.machine1_state.seq;
             if (in_system_state.machine1_state.i = 1) then
                out_system_state.machine1_state.i := 2;
             else
                out_system_state.machine1_state.i := 1;
             end if;
             out_system_state.machine1_state.seq := (((in_system_state.machine1_state.-
seq) + 1)mod 3);
          when (rcv_ack0) =>
              out_system_state.GLOBAL_VARIABLES.ACK := -1;
          when (rcv_ack1) =>
              out_system_state.GLOBAL_VARIABLES.ACK := -1;
          when (snd_ack) =>
              out_system_state.GLOBAL_VARIABLES.ACK := in_system_state.machine2_state.-
exp;
              out_system_state.machine2_state.Rdata := e;
          when (rcv_data) =>
             out_system_state.machine2_state.Rdata  := in_system_state.GLOBAL_VARI-
ABLES.DATA(in_system_state.machine2_state.j);
            out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine2_state.j) :=
E;
            out_system_state.GLOBAL_VARIABLES.SEQ (in_system_state.machine2_state.j) :=
-1;
             if (in_system_state.machine2_state.j = 1) then
                out_system_state.machine2_state.j := 2;
```

```
            else
               out_system_state.machine2_state.j := 1;
            end if;

          out_system_state.machine2_state.exp :=
             (((in_system_state.machine2_state.exp) + 1)mod 3);
        when others =>
           put_line("There is an error in the Action procedure");
    end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(integer'image(tuple.machine2_state.state_number));
   put(" , ");
   put(tuple.machine1_state.seq, width => 1);
   put(" , ");
   put(tuple.machine1_state.i, width => 1);
   put(" , ");
   put(tuple.machine2_state.exp, width => 1);
   put(" , ");
   put(tuple.machine2_state.j, width => 1);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(" ,");
   put(tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(" ]");
end output_Gtuple;


separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(reach,integer'image(tuple.machine2_state.state_number));
   put(reach," , ");
   put(reach,tuple.machine1_state.seq, width => 1);
   put(reach," , ");
   put(reach,tuple.machine1_state.i, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.exp, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.j, width => 1);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(reach," ,");
   put(reach,tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;

separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
      Error_flag := true;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

OUTPUT

REACHABILITY ANALYSIS of : go_back_n_w2

Global State GRAPH

```
 0   [ 0 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , -1 ]        snd_data   1
 1   [ 1 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , -1 ]        snd_data   2
                                                            rcv_data   3
 2   [ 2 , 0 , 2 , 1 , 0 , 1 , D0  0 , D1  1 , -1 ]        rcv_data   4
 4   [ 2 , 1 , 2 , 1 , 1 , 2 , E  -1 , D1  1 , -1 ]        rcv_data   6
 6   [ 2 , 2 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , -1 ]        snd_ack    9
 9   [ 2 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 ,  2 ]        rcv_ack0  12
12   [ 0 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , -1 ]        snd_data  15
15   [ 1 , 0 , 0 , 2 , 2 , 1 , D0  2 , E  -1 , -1 ]        snd_data  18
                                                            rcv_data  19
18   [ 2 , 0 , 1 , 1 , 2 , 1 , D0  2 , D1  0 , -1 ]        rcv_data  22
22   [ 2 , 1 , 1 , 1 , 0 , 2 , E  -1 , D1  0 , -1 ]        rcv_data  26
26   [ 2 , 2 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , -1 ]        snd_ack   29
29   [ 2 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 ,  1 ]        rcv_ack0  32
32   [ 0 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , -1 ]        snd_data  35
35   [ 1 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , -1 ]        snd_data  38
                                                            rcv_data  39
38   [ 2 , 0 , 0 , 1 , 1 , 1 , D0  1 , D1  2 , -1 ]        rcv_data  42
42   [ 2 , 1 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , -1 ]        rcv_data  46
46   [ 2 , 2 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , -1 ]        snd_ack   49
49   [ 2 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 ,  0 ]        rcv_ack0   0
39   [ 1 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , -1 ]        snd_data  42
                                                            snd_ack   43
43   [ 1 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 ,  2 ]        rcv_ack0  47
                                                            snd_data  48
47   [ 0 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , -1 ]        snd_data  50
50   [ 1 , 0 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , -1 ]        snd_data  52
                                                            rcv_data  53
52   [ 2 , 0 , 1 , 2 , 2 , 2 , D0  0 , D1  2 , -1 ]        rcv_data  54
54   [ 2 , 1 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , -1 ]        rcv_data  56
56   [ 2 , 2 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , -1 ]        snd_ack   58
58   [ 2 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 ,  1 ]        rcv_ack0   7
53   [ 1 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , -1 ]        snd_data  54
                                                            snd_ack   55
55   [ 1 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 ,  0 ]        rcv_ack0   0
                                                            snd_data  57
57   [ 2 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 ,  0 ]        rcv_ack1   1
                                                            rcv_data  59
59   [ 2 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 ,  0 ]        rcv_ack1   3
                                                            snd_ack   58
48   [ 2 , 0 , 0 , 1 , 2 , 2 , E  -1 , D1  2 ,  2 ]        rcv_ack1  50
                                                            rcv_data  51
51   [ 2 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 ,  2 ]        rcv_ack1  53
                                                            snd_ack   49
19   [ 1 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , -1 ]        snd_data  22
                                                            snd_ack   23
23   [ 1 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 ,  0 ]        rcv_ack0  27
                                                            snd_data  28
27   [ 0 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , -1 ]        snd_data  30
30   [ 1 , 0 , 1 , 1 , 0 , 2 , E  -1 , D1  0 , -1 ]        snd_data  33
                                                            rcv_data  34
33   [ 2 , 0 , 2 , 2 , 0 , 2 , D0  1 , D1  0 , -1 ]        rcv_data  36
36   [ 2 , 1 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , -1 ]        rcv_data  40
40   [ 2 , 2 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , -1 ]        snd_ack   44
44   [ 2 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 ,  2 ]        rcv_ack0  47
34   [ 1 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , -1 ]        snd_data  36
                                                            snd_ack   37
37   [ 1 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 ,  1 ]        rcv_ack0  32
                                                            snd_data  41
41   [ 2 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 ,  1 ]        rcv_ack1  35
                                                            rcv_data  45
45   [ 2 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 ,  1 ]        rcv_ack1  39
                                                            snd_ack   44
28   [ 2 , 0 , 1 , 1 , 0 , 2 , E  -1 , D1  0 ,  0 ]        rcv_ack1  30
                                                            rcv_data  31
31   [ 2 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 ,  0 ]        rcv_ack1  34
                                                            snd_ack   29
 3   [ 1 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , -1 ]        snd_data   4
                                                            snd_ack    5
```

```
 5   [ 1 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 ,  1 ]
 7   [ 0 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , -1 ]
10   [ 1 , 0 , 2 , 1 , 1 , 2 , E  -1 , D1  1 , -1 ]
13   [ 2 , 0 , 0 , 2 , 1 , 2 , D0  2 , D1  1 , -1 ]
16   [ 2 , 1 , 0 , 2 , 2 , 1 , D0  2 , E  -1 , -1 ]
20   [ 2 , 2 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , -1 ]
24   [ 2 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 ,  0 ]
14   [ 1 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , -1 ]
17   [ 1 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 ,  2 ]
21   [ 2 , 0 , 0 , 2 , 2 , 1 , D0  2 , E  -1 ,  2 ]
25   [ 2 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 ,  2 ]
 8   [ 2 , 0 , 2 , 1 , 1 , 2 , E  -1 , D1  1 ,  1 ]
11   [ 2 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 ,  1 ]
```

```
rcv_ack0   7
snd_data   8
snd_data  10
snd_data  13
rcv_data  14
rcv_data  16
rcv_data  20
snd_ack   24
rcv_ack0  27
snd_data  16
snd_ack   17
rcv_ack0  12
snd_data  21
rcv_ack1  15
rcv_data  25
rcv_ack1  19
snd_ack   24
rcv_ack1  10
rcv_data  11
rcv_ack1  14
snd_ack    9
```

System State GRAPH

```
0   [ 0 , 0 ]    snd_data   [ 1 , 0 ]  1
1   [ 1 , 0 ]    snd_data   [ 2 , 0 ]  2
                 rcv_data   [ 1 , 1 ]  3
2   [ 2 , 0 ]    rcv_data   [ 2 , 1 ]  4
4   [ 2 , 1 ]    rcv_data   [ 2 , 2 ]  6
6   [ 2 , 2 ]    snd_ack    [ 2 , 0 ]  8
8   [ 2 , 0 ]    rcv_ack0   [ 0 , 0 ]  0
3   [ 1 , 1 ]    snd_data   [ 2 , 1 ]  4
                 snd_ack    [ 1 , 0 ]  5
5   [ 1 , 0 ]    rcv_ack0   [ 0 , 0 ]  0
                 snd_data   [ 2 , 0 ]  7
7   [ 2 , 0 ]    rcv_ack1   [ 1 , 0 ]  1
                 rcv_data   [ 2 , 1 ]  9
9   [ 2 , 1 ]    rcv_ack1   [ 1 , 1 ]  3
                 snd_ack    [ 2 , 0 ]  8
```

Machine 1 Array Contents

From	To	Transition	Executed
0	1	snd_data	yes
1	2	snd_data	yes
1	0	rcv_ack0	yes
2	0	rcv_ack0	yes
2	1	rcv_ack1	yes

Machine 2 Array Contents

From	To	Transition	Executed
0	1	rcv_data	yes
1	2	rcv_data	yes
1	0	snd_ack	yes
2	0	snd_ack	yes

APPENDIX G (SCM) GO_BACK_N, W=3

INPUT (FSM)

```
start
machine 1
state 0
trans snd_data 1
state 1
trans rcv_ack0 0
trans snd_data 2
state 2
trans rcv_ack0 0
trans rcv_ack1 1
trans snd_data 3
state 3
trans rcv_ack0 0
trans rcv_ack1 1
trans rcv_ack2 2
machine 2
state 0
trans rcv_data 1
state 1
trans rcv_data 2
trans snd_ack 0
state 2
trans rcv_data 3
trans snd_ack 0
state 3
trans snd_ack 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is


   type scm_transition_type  is (snd_data, rcv_data,
                                 snd_ack,  rcv_ack0,
                                rcv_ack1, rcv_ack2, unused);

   type buffer_type   is (d0,d1,d2,e,a);
   type buffer_array_type is array(1..3) of buffer_type;
   type seq_array_type is array(1..3) of integer range -1..3;

   type machine1_state_type is
      record
          state_number      : natural := 0;
          Sdata             : buffer_array_type := (d0,d1,d2);
          seq               : integer range 0..3 := 0;
          i                 : integer range 1..3 := 1;
   end record;

   type machine2_state_type is
      record
          state_number      : natural := 0;
          Rdata             : buffer_type := e;
          exp               : integer range 0..3 := 0;
          j                 : integer range 1..3 := 1;
   end record;

   type global_variable_type is
      record
          DATA              : buffer_array_type   := (e,e,e);
          SEQ               : seq_array_type      := (-1,-1,-1);
          ACK               : integer range -1..3 := -1;
    end record;

end definitions;
```

PREDICATE-ACTION

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
      temp1 : integer := GLOBAL.ACK + 0;
      temp2 : integer := (GLOBAL.ACK + 1)mod 4;
      temp3 : integer := (GLOBAL.ACK + 2)mod 4;
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.i) = E) and (GLOBAL.SEQ(local.i) = -1)) then
          Push(transition_stack,snd_data);
      end if;
      if ((temp1 = local.seq) and (GLOBAL.ACK /= -1)) then
          Push(transition_stack,rcv_ack0);
      end if;
      if ((temp2 = local.seq) and (GLOBAL.ACK /= -1)) then
          Push(transition_stack,rcv_ack1);
      end if;
      if ((temp3 = local.seq) and (GLOBAL.ACK /= -1)) then
          Push(transition_stack,rcv_ack2);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine1;


separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.j)/=E) and (GLOBAL.SEQ(local.j) = local.exp)) then
          Push(transition_stack,rcv_data);
      end if;
      if (GLOBAL.DATA(local.j)=E) then
         Push(transition_stack,snd_ack);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2; -- this returned value is then checked against the
machine arrays
                                  -- to determine if indeed this transition can be
taken


separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
          when (snd_data) =>
            out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine1_state.i) :=
                in_system_state.machine1_state.Sdata(in_system_state.machine1_state.i);
             out_system_state.GLOBAL_VARIABLES.SEQ(in_system_state.machine1_state.i) :=
                in_system_state.machine1_state.seq;
             begin
             case (in_system_state.machine1_state.i) is
                when 1 =>
                   out_system_state.machine1_state.i := 2;
                when 2 =>
                   out_system_state.machine1_state.i := 3;
                when 3 =>
                   out_system_state.machine1_state.i := 1;
                when others =>
                   null;
             end case;
             end;
              out_system_state.machine1_state.seq := (((in_system_state.machine1_state.-
seq) + 1)mod 4);
          when (rcv_ack0) =>
              out_system_state.GLOBAL_VARIABLES.ACK := -1;
```

```
          when (rcv_ack1) =>
              out_system_state.GLOBAL_VARIABLES.ACK := -1;
          when (rcv_ack2) =>
              out_system_state.GLOBAL_VARIABLES.ACK := -1;
          when (snd_ack) =>
              out_system_state.GLOBAL_VARIABLES.ACK := in_system_state.machine2_state.-
exp;
              out_system_state.machine2_state.Rdata := e;
          when (rcv_data) =>
             out_system_state.machine2_state.Rdata  := in_system_state.GLOBAL_VARI-
ABLES.DATA(in_system_state.machine2_state.j);
            out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine2_state.j) :=
E;
            out_system_state.GLOBAL_VARIABLES.SEQ (in_system_state.machine2_state.j) :=
-1;
             begin
             case (in_system_state.machine2_state.j) is
                when 1 =>
                   out_system_state.machine2_state.j := 2;
                when 2 =>
                   out_system_state.machine2_state.j := 3;
                when 3 =>
                   out_system_state.machine2_state.j := 1;
                when others =>
                   null;
             end case;
             end;
            out_system_state.machine2_state.exp := (((in_system_state.machine2_state.-
exp) + 1)mod 4);
          when others =>
            put_line("There is an error in the Action procedure");
      end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(integer'image(tuple.machine2_state.state_number));
   put(" , ");
   put(tuple.machine1_state.seq, width => 1);
   put(" , ");
   put(tuple.machine1_state.i, width => 1);
   put(" , ");
   put(tuple.machine2_state.exp, width => 1);
   put(" , ");
   put(tuple.machine2_state.j, width => 1);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(3), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(3),width=>2);
   put(" ,");
   put(tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(" ]");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(reach,integer'image(tuple.machine2_state.state_number));
   put(reach," , ");
   put(reach,tuple.machine1_state.seq, width => 1);
   put(reach," , ");
   put(reach,tuple.machine1_state.i, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.exp, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.j, width => 1);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(reach," ,");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(3), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(3),width=>2);
   put(reach," ,");
   put(reach,tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;

separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
```

```
      Error_flag := true;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

OUTPUT

REACHABILITY ANALYSIS of : go_back_n_w3

Global State GRAPH

```
  0   [ 0 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data   1
  1   [ 1 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 , -1 ]      snd_data   2
                                                                   rcv_data   3
  2   [ 2 , 0 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 , -1 ]      snd_data   4
                                                                   rcv_data   5
  4   [ 3 , 0 , 3 , 1 , 0 , 1 , D0  0 , D1  1 , D2  2 , -1 ]      rcv_data   7
  7   [ 3 , 1 , 3 , 1 , 1 , 2 , E  -1 , D1  1 , D2  2 , -1 ]      rcv_data  11
 11   [ 3 , 2 , 3 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 , -1 ]      rcv_data  16
 16   [ 3 , 3 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack    22
 22   [ 3 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack0   29
 29   [ 0 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data   35
 35   [ 1 , 0 , 0 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 , -1 ]      snd_data   43
                                                                   rcv_data   44
 43   [ 2 , 0 , 1 , 3 , 3 , 1 , D0  3 , D1  0 , E  -1 , -1 ]      snd_data   51
                                                                   rcv_data   52
 51   [ 3 , 0 , 2 , 1 , 3 , 1 , D0  3 , D1  0 , D2  1 , -1 ]      rcv_data   61
 61   [ 3 , 1 , 2 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 , -1 ]      rcv_data   68
 68   [ 3 , 2 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 , -1 ]      rcv_data   76
 76   [ 3 , 3 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack    82
 82   [ 3 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  2 ]      rcv_ack0   89
 89   [ 0 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data   95
 95   [ 1 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 , -1 ]      snd_data  103
                                                                   rcv_data  104
103   [ 2 , 0 , 0 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 , -1 ]      snd_data  111
                                                                   rcv_data  112
111   [ 3 , 0 , 1 , 1 , 2 , 1 , D0  2 , D1  3 , D2  0 , -1 ]      rcv_data  121
121   [ 3 , 1 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  0 , -1 ]      rcv_data  128
128   [ 3 , 2 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 , -1 ]      rcv_data  136
136   [ 3 , 3 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack   142
142   [ 3 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  1 ]      rcv_ack0  149
149   [ 0 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data  155
155   [ 1 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 , -1 ]      snd_data  163
                                                                   rcv_data  164
163   [ 2 , 0 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 , -1 ]      snd_data  171
                                                                   rcv_data  172
171   [ 3 , 0 , 0 , 1 , 1 , 1 , D0  1 , D1  2 , D2  3 , -1 ]      rcv_data  181
181   [ 3 , 1 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , D2  3 , -1 ]      rcv_data  188
188   [ 3 , 2 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 , -1 ]      rcv_data  196
196   [ 3 , 3 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack   202
202   [ 3 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack0    0
172   [ 2 , 1 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 , -1 ]      snd_data  181
                                                                   rcv_data  182
182   [ 2 , 2 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data  188
                                                                   snd_ack   189
189   [ 2 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack0  197
                                                                   snd_data  198
197   [ 0 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data  203
203   [ 1 , 0 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 , -1 ]      snd_data  209
                                                                   rcv_data  210
209   [ 2 , 0 , 1 , 2 , 3 , 3 , D0  0 , E  -1 , D2  3 , -1 ]      snd_data  214
                                                                   rcv_data  215
214   [ 3 , 0 , 2 , 3 , 3 , 3 , D0  0 , D1  1 , D2  3 , -1 ]      rcv_data  221
221   [ 3 , 1 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 , -1 ]      rcv_data  227
227   [ 3 , 2 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 , -1 ]      rcv_data  234
234   [ 3 , 3 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack   237
237   [ 3 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,  2 ]      rcv_ack0   17
215   [ 2 , 1 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 , -1 ]      snd_data  221
                                                                   rcv_data  222
222   [ 2 , 2 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data  227
                                                                   snd_ack   228
228   [ 2 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,  1 ]      rcv_ack0    9
                                                                   snd_data  235
235   [ 3 , 0 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,  1 ]      rcv_ack1   13
                                                                   rcv_data  238
238   [ 3 , 1 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,  1 ]      rcv_ack1   20
                                                                   snd_ack   237
210   [ 1 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data  215
                                                                   snd_ack   216
```

```
216   [ 1 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack0   0
                                                                  snd_data 223
223   [ 2 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,  0 ]      rcv_ack1   1
                                                                  snd_data 229
                                                                  rcv_data 230
229   [ 3 , 0 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,  0 ]      rcv_ack2   2
                                                                  rcv_data 236
236   [ 3 , 1 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,  0 ]      rcv_ack2   5
                                                                  rcv_data 239
239   [ 3 , 2 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack2   8
                                                                  snd_ack  237
230   [ 2 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack1   3
                                                                  snd_data 236
                                                                  snd_ack  228
198   [ 3 , 0 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 ,  3 ]      rcv_ack1 203
                                                                  rcv_data 204
204   [ 3 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack1 210
                                                                  snd_ack  202
164   [ 1 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data 172
                                                                  snd_ack  173
173   [ 1 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  2 ]      rcv_ack0 183
                                                                  snd_data 184
183   [ 0 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data 190
190   [ 1 , 0 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 , -1 ]      snd_data 199
                                                                  rcv_data 200
199   [ 2 , 0 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , D2  3 , -1 ]      snd_data 205
                                                                  rcv_data 206
205   [ 3 , 0 , 1 , 2 , 2 , 2 , D0  0 , D1  2 , D2  3 , -1 ]      rcv_data 211
211   [ 3 , 1 , 1 , 2 , 3 , 3 , D0  0 , E  -1 , D2  3 , -1 ]      rcv_data 217
217   [ 3 , 2 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 , -1 ]      rcv_data 224
224   [ 3 , 3 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 , -1 ]      snd_ack  231
231   [ 3 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,  1 ]      rcv_ack0   9
206   [ 2 , 1 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 , -1 ]      snd_data 211
                                                                  rcv_data 212
212   [ 2 , 2 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data 217
                                                                  snd_ack  218
218   [ 2 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack0   0
                                                                  snd_data 225
225   [ 3 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,  0 ]      rcv_ack1   1
                                                                  rcv_data 232
232   [ 3 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack1   3
                                                                  snd_ack  231
200   [ 1 , 1 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data 206
                                                                  snd_ack  207
207   [ 1 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack0 197
                                                                  snd_data 213
213   [ 2 , 0 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 ,  3 ]      rcv_ack1 203
                                                                  snd_data 219
                                                                  rcv_data 220
219   [ 3 , 0 , 1 , 2 , 3 , 3 , D0  0 , E  -1 , D2  3 ,  3 ]      rcv_ack2 209
                                                                  rcv_data 226
226   [ 3 , 1 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,  3 ]      rcv_ack2 215
                                                                  rcv_data 233
233   [ 3 , 2 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack2 222
                                                                  snd_ack  231
220   [ 2 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  3 ]      rcv_ack1 210
                                                                  snd_data 226
                                                                  snd_ack  218
184   [ 2 , 0 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,  2 ]      rcv_ack1 190
                                                                  snd_data 191
                                                                  rcv_data 192
191   [ 3 , 0 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , D2  3 ,  2 ]      rcv_ack2 199
                                                                  rcv_data 201
201   [ 3 , 1 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 ,  2 ]      rcv_ack2 206
                                                                  rcv_data 208
208   [ 3 , 2 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,  2 ]      rcv_ack2 212
                                                                  snd_ack  202
192   [ 2 , 1 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  2 ]      rcv_ack1 200
                                                                  snd_data 201
                                                                  snd_ack  189
112   [ 2 , 1 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 , -1 ]      snd_data 121
                                                                  rcv_data 122
122   [ 2 , 2 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 , -1 ]      snd_data 128
                                                                  snd_ack  129
129   [ 2 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]      rcv_ack0 137
                                                                  snd_data 138
```

```
137   [ 0 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 143
143   [ 1 , 0 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 , -1 ]        snd_data 150
                                                                    rcv_data 151
150   [ 2 , 0 , 2 , 2 , 0 , 3 , D0  1 , E  -1 , D2  0 , -1 ]        snd_data 156
                                                                    rcv_data 157
156   [ 3 , 0 , 3 , 3 , 0 , 3 , D0  1 , D1  2 , D2  0 , -1 ]        rcv_data 165
165   [ 3 , 1 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 , -1 ]        rcv_data 174
174   [ 3 , 2 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 , -1 ]        rcv_data 185
185   [ 3 , 3 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 , -1 ]        snd_ack  193
193   [ 3 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]        rcv_ack0 197
157   [ 2 , 1 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 , -1 ]        snd_data 165
                                                                    rcv_data 166
166   [ 2 , 2 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 174
                                                                    snd_ack  175
175   [ 2 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  2 ]        rcv_ack0 183
                                                                    snd_data 186
186   [ 3 , 0 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,  2 ]        rcv_ack1 190
                                                                    rcv_data 194
194   [ 3 , 1 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  2 ]        rcv_ack1 200
                                                                    snd_ack  193
151   [ 1 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 157
                                                                    snd_ack  158
158   [ 1 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  1 ]        rcv_ack0 149
                                                                    snd_data 167
167   [ 2 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,  1 ]        rcv_ack1 155
                                                                    snd_data 176
                                                                    rcv_data 177
176   [ 3 , 0 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 ,  1 ]        rcv_ack2 163
                                                                    rcv_data 187
187   [ 3 , 1 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,  1 ]        rcv_ack2 172
                                                                    rcv_data 195
195   [ 3 , 2 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,  1 ]        rcv_ack2 182
                                                                    snd_ack  193
177   [ 2 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  1 ]        rcv_ack1 164
                                                                    snd_data 187
                                                                    snd_ack  175
138   [ 3 , 0 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 ,  0 ]        rcv_ack1 143
                                                                    rcv_data 144
144   [ 3 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]        rcv_ack1 151
                                                                    snd_ack  142
104   [ 1 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 112
                                                                    snd_ack  113
113   [ 1 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  3 ]        rcv_ack0 123
                                                                    snd_data 124
123   [ 0 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 130
130   [ 1 , 0 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 , -1 ]        snd_data 139
                                                                    rcv_data 140
139   [ 2 , 0 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  0 , -1 ]        snd_data 145
                                                                    rcv_data 146
145   [ 3 , 0 , 2 , 2 , 3 , 2 , D0  1 , D1  3 , D2  0 , -1 ]        rcv_data 152
152   [ 3 , 1 , 2 , 2 , 0 , 3 , D0  1 , E  -1 , D2  0 , -1 ]        rcv_data 159
159   [ 3 , 2 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 , -1 ]        rcv_data 168
168   [ 3 , 3 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 , -1 ]        snd_ack  178
178   [ 3 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  2 ]        rcv_ack0 183
146   [ 2 , 1 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 , -1 ]        snd_data 152
                                                                    rcv_data 153
153   [ 2 , 2 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 159
                                                                    snd_ack  160
160   [ 2 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  1 ]        rcv_ack0 149
                                                                    snd_data 169
169   [ 3 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,  1 ]        rcv_ack1 155
                                                                    rcv_data 179
179   [ 3 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  1 ]        rcv_ack1 164
                                                                    snd_ack  178
140   [ 1 , 1 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 , -1 ]        snd_data 146
                                                                    snd_ack  147
147   [ 1 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]        rcv_ack0 137
                                                                    snd_data 154
154   [ 2 , 0 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 ,  0 ]        rcv_ack1 143
                                                                    snd_data 161
                                                                    rcv_data 162
161   [ 3 , 0 , 2 , 2 , 0 , 3 , D0  1 , E  -1 , D2  0 ,  0 ]        rcv_ack2 150
                                                                    rcv_data 170
170   [ 3 , 1 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,  0 ]        rcv_ack2 157
                                                                    rcv_data 180
180   [ 3 , 2 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,  0 ]        rcv_ack2 166
```

```
                                                                                snd_ack  178
162   [ 2 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]                   rcv_ack1 151
                                                                                snd_data 170
                                                                                snd_ack  160
124   [ 2 , 0 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,  3 ]                   rcv_ack1 130
                                                                                snd_data 131
                                                                                rcv_data 132
131   [ 3 , 0 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  0 ,  3 ]                   rcv_ack2 139
                                                                                rcv_data 141
141   [ 3 , 1 , 1 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 ,  3 ]                   rcv_ack2 146
                                                                                rcv_data 148
148   [ 3 , 2 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,  3 ]                   rcv_ack2 153
                                                                                snd_ack  142
132   [ 2 , 1 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]                   rcv_ack1 140
                                                                                snd_data 141
                                                                                snd_ack  129
 52   [ 2 , 1 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 , -1 ]                   snd_data  61
                                                                                rcv_data  62
 62   [ 2 , 2 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data  68
                                                                                snd_ack   69
 69   [ 2 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  1 ]                   rcv_ack0  77
                                                                                snd_data  78
 77   [ 0 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data  83
 83   [ 1 , 0 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 , -1 ]                   snd_data  90
                                                                                rcv_data  91
 90   [ 2 , 0 , 3 , 2 , 1 , 3 , D0  2 , E  -1 , D2  1 , -1 ]                   snd_data  96
                                                                                rcv_data  97
 96   [ 3 , 0 , 0 , 3 , 1 , 3 , D0  2 , D1  3 , D2  1 , -1 ]                   rcv_data 105
105   [ 3 , 1 , 0 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 , -1 ]                   rcv_data 114
114   [ 3 , 2 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 , -1 ]                   rcv_data 125
125   [ 3 , 3 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_ack  133
133   [ 3 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]                   rcv_ack0 137
 97   [ 2 , 1 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 , -1 ]                   snd_data 105
                                                                                rcv_data 106
106   [ 2 , 2 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data 114
                                                                                snd_ack  115
115   [ 2 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  3 ]                   rcv_ack0 123
                                                                                snd_data 126
126   [ 3 , 0 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,  3 ]                   rcv_ack1 130
                                                                                rcv_data 134
134   [ 3 , 1 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]                   rcv_ack1 140
                                                                                snd_ack  133
 91   [ 1 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data  97
                                                                                snd_ack   98
 98   [ 1 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  2 ]                   rcv_ack0  89
                                                                                snd_data 107
107   [ 2 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,  2 ]                   rcv_ack1  95
                                                                                snd_data 116
                                                                                rcv_data 117
116   [ 3 , 0 , 0 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,  2 ]                   rcv_ack2 103
                                                                                rcv_data 127
127   [ 3 , 1 , 0 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,  2 ]                   rcv_ack2 112
                                                                                rcv_data 135
135   [ 3 , 2 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,  2 ]                   rcv_ack2 122
                                                                                snd_ack  133
117   [ 2 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  2 ]                   rcv_ack1 104
                                                                                snd_data 127
                                                                                snd_ack  115
 78   [ 3 , 0 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,  1 ]                   rcv_ack1  83
                                                                                rcv_data  84
 84   [ 3 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  1 ]                   rcv_ack1  91
                                                                                snd_ack   82
 44   [ 1 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data  52
                                                                                snd_ack   53
 53   [ 1 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,  0 ]                   rcv_ack0  63
                                                                                snd_data  64
 63   [ 0 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_data  70
 70   [ 1 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 , -1 ]                   snd_data  79
                                                                                rcv_data  80
 79   [ 2 , 0 , 2 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 , -1 ]                   snd_data  85
                                                                                rcv_data  86
 85   [ 3 , 0 , 3 , 2 , 0 , 2 , D0  2 , D1  0 , D2  1 , -1 ]                   rcv_data  92
 92   [ 3 , 1 , 3 , 2 , 1 , 3 , D0  2 , E  -1 , D2  1 , -1 ]                   rcv_data  99
 99   [ 3 , 2 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 , -1 ]                   rcv_data 108
108   [ 3 , 3 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 , -1 ]                   snd_ack  118
118   [ 3 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  3 ]                   rcv_ack0 123
```

```
 86   [ 2 , 1 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 , -1 ]        snd_data  92
                                                                    rcv_data  93
 93   [ 2 , 2 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  99
                                                                    snd_ack  100
100   [ 2 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  2 ]       rcv_ack0  89
                                                                    snd_data 109
109   [ 3 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,  2 ]       rcv_ack1  95
                                                                    rcv_data 119
119   [ 3 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  2 ]       rcv_ack1 104
                                                                    snd_ack  118
 80   [ 1 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  86
                                                                    snd_ack   87
 87   [ 1 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  1 ]       rcv_ack0  77
                                                                    snd_data  94
 94   [ 2 , 0 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,  1 ]       rcv_ack1  83
                                                                    snd_data 101
                                                                    rcv_data 102
101   [ 3 , 0 , 3 , 2 , 1 , 3 , D0  2 , E  -1 , D2  1 ,  1 ]       rcv_ack2  90
                                                                    rcv_data 110
110   [ 3 , 1 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,  1 ]       rcv_ack2  97
                                                                    rcv_data 120
120   [ 3 , 2 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,  1 ]       rcv_ack2 106
                                                                    snd_ack  118
102   [ 2 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  1 ]       rcv_ack1  91
                                                                    snd_data 110
                                                                    snd_ack  100
 64   [ 2 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,  0 ]       rcv_ack1  70
                                                                    snd_data  71
                                                                    rcv_data  72
 71   [ 3 , 0 , 2 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 ,  0 ]       rcv_ack2  79
                                                                    rcv_data  81
 81   [ 3 , 1 , 2 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,  0 ]       rcv_ack2  86
                                                                    rcv_data  88
 88   [ 3 , 2 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,  0 ]       rcv_ack2  93
                                                                    snd_ack   82
 72   [ 2 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]       rcv_ack1  80
                                                                    snd_data  81
                                                                    snd_ack   69
  5   [ 2 , 1 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 , -1 ]       snd_data   7
                                                                    rcv_data   8
  8   [ 2 , 2 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  11
                                                                    snd_ack   12
 12   [ 2 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,  2 ]       rcv_ack0  17
                                                                    snd_data  18
 17   [ 0 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  23
 23   [ 1 , 0 , 3 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 , -1 ]       snd_data  30
                                                                    rcv_data  31
 30   [ 2 , 0 , 0 , 2 , 2 , 3 , D0  3 , E  -1 , D2  2 , -1 ]       snd_data  36
                                                                    rcv_data  37
 36   [ 3 , 0 , 1 , 3 , 2 , 3 , D0  3 , D1  0 , D2  2 , -1 ]       rcv_data  45
 45   [ 3 , 1 , 1 , 3 , 3 , 1 , D0  3 , D1  0 , E  -1 , -1 ]       rcv_data  54
 54   [ 3 , 2 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 , -1 ]       rcv_data  65
 65   [ 3 , 3 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 , -1 ]       snd_ack   73
 73   [ 3 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  1 ]       rcv_ack0  77
 37   [ 2 , 1 , 0 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 , -1 ]       snd_data  45
                                                                    rcv_data  46
 46   [ 2 , 2 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  54
                                                                    snd_ack   55
 55   [ 2 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,  0 ]       rcv_ack0  63
                                                                    snd_data  66
 66   [ 3 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,  0 ]       rcv_ack1  70
                                                                    rcv_data  74
 74   [ 3 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  0 ]       rcv_ack1  80
                                                                    snd_ack   73
 31   [ 1 , 1 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 , -1 ]       snd_data  37
                                                                    snd_ack   38
 38   [ 1 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,  3 ]       rcv_ack0  29
                                                                    snd_data  47
 47   [ 2 , 0 , 0 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,  3 ]       rcv_ack1  35
                                                                    snd_data  56
                                                                    rcv_data  57
 56   [ 3 , 0 , 1 , 3 , 3 , 1 , D0  3 , D1  0 , E  -1 ,  3 ]       rcv_ack2  43
                                                                    rcv_data  67
 67   [ 3 , 1 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,  3 ]       rcv_ack2  52
                                                                    rcv_data  75
 75   [ 3 , 2 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,  3 ]       rcv_ack2  62
```

		snd_ack	73
57	[2 , 1 , 0 , 2 , 0 , 2 , E -1 , E -1 , E -1 , 3]	rcv_ack1	44
		snd_data	67
		snd_ack	55
18	[3 , 0 , 3 , 1 , 2 , 3 , E -1 , E -1 , D2 2 , 2]	rcv_ack1	23
		rcv_data	24
24	[3 , 1 , 3 , 1 , 3 , 1 , E -1 , E -1 , E -1 , 2]	rcv_ack1	31
		snd_ack	22
3	[1 , 1 , 1 , 2 , 1 , 2 , E -1 , E -1 , E -1 , -1]	snd_data	5
		snd_ack	6
6	[1 , 0 , 1 , 2 , 1 , 2 , E -1 , E -1 , E -1 , 1]	rcv_ack0	9
		snd_data	10
9	[0 , 0 , 1 , 2 , 1 , 2 , E -1 , E -1 , E -1 , -1]	snd_data	13
13	[1 , 0 , 2 , 3 , 1 , 2 , E -1 , D1 1 , E -1 , -1]	snd_data	19
		rcv_data	20
19	[2 , 0 , 3 , 1 , 1 , 2 , E -1 , D1 1 , D2 2 , -1]	snd_data	25
		rcv_data	26
25	[3 , 0 , 0 , 2 , 1 , 2 , D0 3 , D1 1 , D2 2 , -1]	rcv_data	32
32	[3 , 1 , 0 , 2 , 2 , 3 , D0 3 , E -1 , D2 2 , -1]	rcv_data	39
39	[3 , 2 , 0 , 2 , 3 , 1 , D0 3 , E -1 , E -1 , -1]	rcv_data	48
48	[3 , 3 , 0 , 2 , 0 , 2 , E -1 , E -1 , E -1 , -1]	snd_ack	58
58	[3 , 0 , 0 , 2 , 0 , 2 , E -1 , E -1 , E -1 , 0]	rcv_ack0	63
26	[2 , 1 , 3 , 1 , 2 , 3 , E -1 , E -1 , D2 2 , -1]	snd_data	32
		rcv_data	33
33	[2 , 2 , 3 , 1 , 3 , 1 , E -1 , E -1 , E -1 , -1]	snd_data	39
		snd_ack	40
40	[2 , 0 , 3 , 1 , 3 , 1 , E -1 , E -1 , E -1 , 3]	rcv_ack0	29
		snd_data	49
49	[3 , 0 , 0 , 2 , 3 , 1 , D0 3 , E -1 , E -1 , 3]	rcv_ack1	35
		rcv_data	59
59	[3 , 1 , 0 , 2 , 0 , 2 , E -1 , E -1 , E -1 , 3]	rcv_ack1	44
		snd_ack	58
20	[1 , 1 , 2 , 3 , 2 , 3 , E -1 , E -1 , E -1 , -1]	snd_data	26
		snd_ack	27
27	[1 , 0 , 2 , 3 , 2 , 3 , E -1 , E -1 , E -1 , 2]	rcv_ack0	17
		snd_data	34
34	[2 , 0 , 3 , 1 , 2 , 3 , E -1 , E -1 , D2 2 , 2]	rcv_ack1	23
		snd_data	41
		rcv_data	42
41	[3 , 0 , 0 , 2 , 2 , 3 , D0 3 , E -1 , D2 2 , 2]	rcv_ack2	30
		rcv_data	50
50	[3 , 1 , 0 , 2 , 3 , 1 , D0 3 , E -1 , E -1 , 2]	rcv_ack2	37
		rcv_data	60
60	[3 , 2 , 0 , 2 , 0 , 2 , E -1 , E -1 , E -1 , 2]	rcv_ack2	46
		snd_ack	58
42	[2 , 1 , 3 , 1 , 3 , 1 , E -1 , E -1 , E -1 , 2]	rcv_ack1	31
		snd_data	50
		snd_ack	40
10	[2 , 0 , 2 , 3 , 1 , 2 , E -1 , D1 1 , E -1 , 1]	rcv_ack1	13
		snd_data	14
		rcv_data	15
14	[3 , 0 , 3 , 1 , 1 , 2 , E -1 , D1 1 , D2 2 , 1]	rcv_ack2	19
		rcv_data	21
21	[3 , 1 , 3 , 1 , 2 , 3 , E -1 , E -1 , D2 2 , 1]	rcv_ack2	26
		rcv_data	28
28	[3 , 2 , 3 , 1 , 3 , 1 , E -1 , E -1 , E -1 , 1]	rcv_ack2	33
		snd_ack	22
15	[2 , 1 , 2 , 3 , 2 , 3 , E -1 , E -1 , E -1 , 1]	rcv_ack1	20
		snd_data	21
		snd_ack	12

System State GRAPH

0	[0 , 0]	snd_data	[1 , 0]	1
1	[1 , 0]	snd_data	[2 , 0]	2
		rcv_data	[1 , 1]	3
2	[2 , 0]	snd_data	[3 , 0]	4
		rcv_data	[2 , 1]	5
4	[3 , 0]	rcv_data	[3 , 1]	7
7	[3 , 1]	rcv_data	[3 , 2]	10
10	[3 , 2]	rcv_data	[3 , 3]	14
14	[3 , 3]	snd_ack	[3 , 0]	17
17	[3 , 0]	rcv_ack0	[0 , 0]	0
5	[2 , 1]	snd_data	[3 , 1]	7

```
                    rcv_data    [ 2 , 2 ]   8
 8    [ 2 , 2 ]     snd_data    [ 3 , 2 ]   10
                    snd_ack     [ 2 , 0 ]   11
11    [ 2 , 0 ]     rcv_ack0    [ 0 , 0 ]   0
                    snd_data    [ 3 , 0 ]   15
15    [ 3 , 0 ]     rcv_ack1    [ 1 , 0 ]   1
                    rcv_data    [ 3 , 1 ]   18
18    [ 3 , 1 ]     rcv_ack1    [ 1 , 1 ]   3
                    snd_ack     [ 3 , 0 ]   17
 3    [ 1 , 1 ]     snd_data    [ 2 , 1 ]   5
                    snd_ack     [ 1 , 0 ]   6
 6    [ 1 , 0 ]     rcv_ack0    [ 0 , 0 ]   0
                    snd_data    [ 2 , 0 ]   9
 9    [ 2 , 0 ]     rcv_ack1    [ 1 , 0 ]   1
                    snd_data    [ 3 , 0 ]   12
                    rcv_data    [ 2 , 1 ]   13
12    [ 3 , 0 ]     rcv_ack2    [ 2 , 0 ]   2
                    rcv_data    [ 3 , 1 ]   16
16    [ 3 , 1 ]     rcv_ack2    [ 2 , 1 ]   5
                    rcv_data    [ 3 , 2 ]   19
19    [ 3 , 2 ]     rcv_ack2    [ 2 , 2 ]   8
                    snd_ack     [ 3 , 0 ]   17
13    [ 2 , 1 ]     rcv_ack1    [ 1 , 1 ]   3
                    snd_data    [ 3 , 1 ]   16
                    snd_ack     [ 2 , 0 ]   11
```

Machine 1 Array Contents			
From	To	Transition	Executed
0	1	snd_data	yes
1	0	rcv_ack0	yes
1	2	snd_data	yes
2	0	rcv_ack0	yes
2	1	rcv_ack1	yes
2	3	snd_data	yes
3	0	rcv_ack0	yes
3	1	rcv_ack1	yes
3	2	rcv_ack2	yes

Machine 2 Array Contents			
From	To	Transition	Executed
0	1	rcv_data	yes
1	2	rcv_data	yes
1	0	snd_ack	yes
2	3	rcv_data	yes
2	0	snd_ack	yes
3	0	snd_ack	yes

APPENDIX H (SCM) GO_BACK_N, W=4

INPUT (FSM)

```
start
machine 1
state 0
trans snd_data 1
state 1
trans rcv_ack0 0
trans snd_data 2
state 2
trans rcv_ack0 0
trans rcv_ack1 1
trans snd_data 3
state 3
trans rcv_ack0 0
trans rcv_ack1 1
trans rcv_ack2 2
trans snd_data 4
state 4
trans rcv_ack0 0
trans rcv_ack1 1
trans rcv_ack2 2
trans rcv_ack3 3
machine 2
state 0
trans rcv_data 1
state 1
trans rcv_data 2
trans snd_ack 0
state 2
trans rcv_data 3
trans snd_ack 0
state 3
trans rcv_data 4
trans snd_ack 0
state 4
trans snd_ack 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is

   type scm_transition_type  is (snd_data, rcv_data,
                                 snd_ack,  rcv_ack0,
                                  rcv_ack1, rcv_ack2,
                                 rcv_ack3, unused);

   type buffer_type   is (d0,d1,d2,d3,e,a);
   type buffer_array_type is array(1..4) of buffer_type;
   type seq_array_type is array(1..4) of integer range -1..4;

   type machine1_state_type is
      record
         state_number     : natural := 0;
         Sdata            : buffer_array_type := (d0,d1,d2,d3);
         seq              : integer range 0..4 := 0;
         i                : integer range 1..4 := 1;
   end record;

   type machine2_state_type is
      record
         state_number     : natural := 0;
         Rdata            : buffer_type := e;
         exp              : integer range 0..4 := 0;
         j                : integer range 1..4 := 1;
   end record;

   type global_variable_type is
      record
         DATA             : buffer_array_type   := (e,e,e,e);
         SEQ              : seq_array_type      := (-1,-1,-1,-1);
         ACK              : integer range -1..4 := -1;
    end record;

end definitions;
```

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
      temp1 : integer := GLOBAL.ACK + 0;
      temp2 : integer := (GLOBAL.ACK + 1)mod 5;
      temp3 : integer := (GLOBAL.ACK + 2)mod 5;
      temp4 : integer := (GLOBAL.ACK + 3)mod 5;
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.i) = E) and (GLOBAL.SEQ(local.i) = -1)) then
         Push(transition_stack,snd_data);
      end if;
      if ((temp1 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack0);
      end if;
      if ((temp2 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack1);
      end if;
      if ((temp3 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack2);
      end if;
      if ((temp4 = local.seq) and (GLOBAL.ACK /= -1)) then
         Push(transition_stack,rcv_ack3);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine1;


separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                     GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA(local.j)/=E) and (GLOBAL.SEQ(local.j) = local.exp)) then
         Push(transition_stack,rcv_data);
      end if;
      if (GLOBAL.DATA(local.j)=E) then
        Push(transition_stack,snd_ack);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2;

separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition: in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
         when (snd_data) =>
           out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine1_state.i) :=
              in_system_state.machine1_state.Sdata(in_system_state.machine1_state.i);
            out_system_state.GLOBAL_VARIABLES.SEQ(in_system_state.machine1_state.i) :=
               in_system_state.machine1_state.seq;
             begin
             case (in_system_state.machine1_state.i) is
                when 1 =>
                   out_system_state.machine1_state.i := 2;
                when 2 =>
                   out_system_state.machine1_state.i := 3;
                when 3 =>
                   out_system_state.machine1_state.i := 4;
                when 4 =>
                   out_system_state.machine1_state.i := 1;
                when others =>
                   null;
             end case;
             end;
              out_system_state.machine1_state.seq :=
                 (((in_system_state.machine1_state.seq) + 1)mod 5);
```

```
            when (rcv_ack0) =>
                out_system_state.GLOBAL_VARIABLES.ACK := -1;
            when (rcv_ack1) =>
                out_system_state.GLOBAL_VARIABLES.ACK := -1;
            when (rcv_ack2) =>
                out_system_state.GLOBAL_VARIABLES.ACK := -1;
           when (rcv_ack3) =>
               out_system_state.GLOBAL_VARIABLES.ACK := -1;
            when (snd_ack) =>
                out_system_state.GLOBAL_VARIABLES.ACK := in_system_state.machine2_state.-
exp;
                out_system_state.machine2_state.Rdata := e;
            when (rcv_data) =>
               out_system_state.machine2_state.Rdata  :=
                  in_system_state.GLOBAL_VARIABLES.
                  DATA(in_system_state.machine2_state.j);
              out_system_state.GLOBAL_VARIABLES.DATA(in_system_state.machine2_state.j) :=
                  E;
              out_system_state.GLOBAL_VARIABLES.SEQ (in_system_state.machine2_state.j) :=
                  -1;
               begin
               case (in_system_state.machine2_state.j) is
                  when 1 =>
                     out_system_state.machine2_state.j := 2;
                  when 2 =>
                     out_system_state.machine2_state.j := 3;
                  when 3 =>
                     out_system_state.machine2_state.j := 4;
                  when 4 =>
                     out_system_state.machine2_state.j := 1;
                  when others =>
                     null;
               end case;
               end;
             out_system_state.machine2_state.exp := (((in_system_state.machine2_state.
                exp) + 1)mod 5);
            when others =>
               put_line("There is an error in the Action procedure");
      end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(integer'image(tuple.machine2_state.state_number));
   put(" , ");
   put(tuple.machine1_state.seq, width => 1);
   put(" , ");
   put(tuple.machine1_state.i, width => 1);
   put(" , ");
   put(tuple.machine2_state.exp, width => 1);
   put(" , ");
   put(tuple.machine2_state.j, width => 1);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA(3), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(3),width=>2);
   put(" ,");
   put(tuple.GLOBAL_VARIABLES.DATA(4), width =>3);
   put(tuple.GLOBAL_VARIABLES.SEQ(4),width=>2);
   put(" ,");
   put(tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(" ]");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number) & " ,");
   put(reach,integer'image(tuple.machine2_state.state_number));
   put(reach," , ");
   put(reach,tuple.machine1_state.seq, width => 1);
   put(reach," , ");
   put(reach,tuple.machine1_state.i, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.exp, width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.j, width => 1);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(1), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(1),width=>2);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(2), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(2),width=>2);
   put(reach," ,");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(3), width =>3);
   put(reach,tuple.GLOBAL_VARIABLES.SEQ(3),width=>2);
   put(reach," ,");
   put(reach,tuple.GLOBAL_VARIABLES.ACK, width => 3);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;

separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
```

```
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
      Error_flag := true;
   else
      Error_flag := false;
  end if;
end output_Gstate_node;
```

OUTPUT

REACHABILITY ANALYSIS of : go_back_n_w4

Global State GRAPH

```
  0   [ 0 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   1
  1   [ 1 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   2
                                                                    rcv_data   3
  2   [ 2 , 0 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 , -1 ] snd_data   4
                                                                    rcv_data   5
  4   [ 3 , 0 , 3 , 4 , 0 , 1 , D0  0 , D1  1 , D2  2 ,E  -1 , -1 ] snd_data   7
                                                                    rcv_data   8
  7   [ 4 , 0 , 4 , 1 , 0 , 1 , D0  0 , D1  1 , D2  2 ,D3  3 , -1 ] rcv_data  12
 12   [ 4 , 1 , 4 , 1 , 1 , 2 , E  -1 , D1  1 , D2  2 ,D3  3 , -1 ] rcv_data  18
 18   [ 4 , 2 , 4 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 ,D3  3 , -1 ] rcv_data  26
 26   [ 4 , 3 , 4 , 1 , 3 , 4 , E  -1 , E  -1 , E  -1 ,D3  3 , -1 ] rcv_data  36
 36   [ 4 , 4 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack    47
 47   [ 4 , 0 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0  60
 60   [ 0 , 0 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   72
 72   [ 1 , 0 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   87
                                                                    rcv_data   88
 87   [ 2 , 0 , 1 , 3 , 4 , 1 , D0  4 , D1  0 , E  -1 ,E  -1 , -1 ] snd_data  102
                                                                    rcv_data  103
102   [ 3 , 0 , 2 , 4 , 4 , 1 , D0  4 , D1  0 , D2  1 ,E  -1 , -1 ] snd_data  120
                                                                    rcv_data  121
120   [ 4 , 0 , 3 , 1 , 4 , 1 , D0  4 , D1  0 , D2  1 ,D3  2 , -1 ] rcv_data  134
134   [ 4 , 1 , 3 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 ,D3  2 , -1 ] rcv_data  150
150   [ 4 , 2 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 , -1 ] rcv_data  162
162   [ 4 , 3 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 , -1 ] rcv_data  176
176   [ 4 , 4 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack    187
187   [ 4 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0  200
200   [ 0 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   212
212   [ 1 , 0 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   227
                                                                    rcv_data   228
227   [ 2 , 0 , 0 , 3 , 3 , 1 , D0  3 , D1  4 , E  -1 ,E  -1 , -1 ] snd_data   242
                                                                    rcv_data   243
242   [ 3 , 0 , 1 , 4 , 3 , 1 , D0  3 , D1  4 , D2  0 ,E  -1 , -1 ] snd_data   260
                                                                    rcv_data   261
260   [ 4 , 0 , 2 , 1 , 3 , 1 , D0  3 , D1  4 , D2  0 ,D3  1 , -1 ] rcv_data   274
274   [ 4 , 1 , 2 , 1 , 4 , 2 , E  -1 , D1  4 , D2  0 ,D3  1 , -1 ] rcv_data   290
290   [ 4 , 2 , 2 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 ,D3  1 , -1 ] rcv_data   302
302   [ 4 , 3 , 2 , 1 , 1 , 4 , E  -1 , E  -1 , E  -1 ,D3  1 , -1 ] rcv_data   316
316   [ 4 , 4 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack    327
327   [ 4 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0  340
340   [ 0 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   352
352   [ 1 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   367
                                                                    rcv_data   368
367   [ 2 , 0 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 , -1 ] snd_data   382
                                                                    rcv_data   383
382   [ 3 , 0 , 0 , 4 , 2 , 1 , D0  2 , D1  3 , D2  4 ,E  -1 , -1 ] snd_data   400
                                                                    rcv_data   401
400   [ 4 , 0 , 1 , 1 , 2 , 1 , D0  2 , D1  3 , D2  4 ,D3  0 , -1 ] rcv_data   414
414   [ 4 , 1 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  4 ,D3  0 , -1 ] rcv_data   430
430   [ 4 , 2 , 1 , 1 , 4 , 3 , E  -1 , E  -1 , D2  4 ,D3  0 , -1 ] rcv_data   442
442   [ 4 , 3 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 , -1 ] rcv_data   456
456   [ 4 , 4 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack    467
467   [ 4 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0  480
480   [ 0 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   492
492   [ 1 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   507
                                                                    rcv_data   508
507   [ 2 , 0 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 ,E  -1 , -1 ] snd_data   522
                                                                    rcv_data   523
522   [ 3 , 0 , 4 , 4 , 1 , 1 , D0  1 , D1  2 , D2  3 ,E  -1 , -1 ] snd_data   540
                                                                    rcv_data   541
540   [ 4 , 0 , 0 , 1 , 1 , 1 , D0  1 , D1  2 , D2  3 ,D3  4 , -1 ] rcv_data   554
554   [ 4 , 1 , 0 , 1 , 2 , 2 , E  -1 , D1  2 , D2  3 ,D3  4 , -1 ] rcv_data   570
570   [ 4 , 2 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 ,D3  4 , -1 ] rcv_data   582
582   [ 4 , 3 , 0 , 1 , 4 , 4 , E  -1 , E  -1 , E  -1 ,D3  4 , -1 ] rcv_data   596
596   [ 4 , 4 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack    607
607   [ 4 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0    0
541   [ 3 , 1 , 4 , 4 , 2 , 2 , E  -1 , D1  2 , D2  3 ,E  -1 , -1 ] snd_data   554
                                                                    rcv_data   555
555   [ 3 , 2 , 4 , 4 , 3 , 3 , E  -1 , E  -1 , D2  3 ,E  -1 , -1 ] snd_data   570
                                                                    rcv_data   571
```

```
571  [ 3 , 3 , 4 , 4 , 4 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 582
                                                                   snd_ack  583
583  [ 3 , 0 , 4 , 4 , 4 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 597
                                                                   snd_data 598
597  [ 0 , 0 , 4 , 4 , 4 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 608
608  [ 1 , 0 , 0 , 1 , 4 , 4 , E  -1 , E  -1 , E  -1 ,D3  4 , -1 ] snd_data 620
                                                                   rcv_data 621
620  [ 2 , 0 , 1 , 2 , 4 , 4 , D0  0 , E  -1 , E  -1 ,D3  4 , -1 ] snd_data 631
                                                                   rcv_data 632
631  [ 3 , 0 , 2 , 3 , 4 , 4 , D0  0 , D1  1 , E  -1 ,D3  4 , -1 ] snd_data 645
                                                                   rcv_data 646
645  [ 4 , 0 , 3 , 4 , 4 , 4 , D0  0 , D1  1 , D2  2 ,D3  4 , -1 ] rcv_data 658
658  [ 4 , 1 , 3 , 4 , 0 , 1 , D0  0 , D1  1 , D2  2 ,E  -1 , -1 ] rcv_data 673
673  [ 4 , 2 , 3 , 4 , 1 , 2 , E  -1 , D1  1 , D2  2 ,E  -1 , -1 ] rcv_data 682
682  [ 4 , 3 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 , -1 ] rcv_data 692
692  [ 4 , 4 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  696
696  [ 4 , 0 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0  37
646  [ 3 , 1 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 , -1 ] snd_data 658
                                                                   rcv_data 659
659  [ 3 , 2 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 , -1 ] snd_data 673
                                                                   rcv_data 674
674  [ 3 , 3 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 682
                                                                   snd_ack  683
683  [ 3 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0  20
                                                                   snd_data 693
693  [ 4 , 0 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 ,  2 ] rcv_ack1  28
                                                                   rcv_data 697
697  [ 4 , 1 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1  40
                                                                   snd_ack  696
632  [ 2 , 1 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 646
                                                                   rcv_data 647
647  [ 2 , 2 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 659
                                                                   snd_ack  660
660  [ 2 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0  10
                                                                   snd_data 675
675  [ 3 , 0 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 ,  1 ] rcv_ack1  15
                                                                   snd_data 684
                                                                   rcv_data 685
684  [ 4 , 0 , 3 , 4 , 1 , 2 , E  -1 , D1  1 , D2  2 ,E  -1 ,  1 ] rcv_ack2  22
                                                                   rcv_data 694
694  [ 4 , 1 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 ,  1 ] rcv_ack2  32
                                                                   rcv_data 698
698  [ 4 , 2 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2  44
                                                                   snd_ack  696
685  [ 3 , 1 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1  23
                                                                   snd_data 694
                                                                   snd_ack  683
621  [ 1 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 632
                                                                   snd_ack  633
633  [ 1 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0   0
                                                                   snd_data 648
648  [ 2 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1   1
                                                                   snd_data 661
                                                                   rcv_data 662
661  [ 3 , 0 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   2
                                                                   snd_data 676
                                                                   rcv_data 677
676  [ 4 , 0 , 3 , 4 , 0 , 1 , D0  0 , D1  1 , D2  2 ,E  -1 ,  0 ] rcv_ack3   4
                                                                   rcv_data 686
686  [ 4 , 1 , 3 , 4 , 1 , 2 , E  -1 , D1  1 , D2  2 ,E  -1 ,  0 ] rcv_ack3   8
                                                                   rcv_data 695
695  [ 4 , 2 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 ,  0 ] rcv_ack3  13
                                                                   rcv_data 699
699  [ 4 , 3 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack3  19
                                                                   snd_ack  696
677  [ 3 , 1 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   5
                                                                   snd_data 686
                                                                   rcv_data 687
687  [ 3 , 2 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   9
                                                                   snd_data 695
                                                                   snd_ack  683
662  [ 2 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1   3
                                                                   snd_data 677
                                                                   snd_ack  660
598  [ 4 , 0 , 0 , 1 , 4 , 4 , E  -1 , E  -1 , E  -1 ,D3  4 ,  4 ] rcv_ack1 608
                                                                   rcv_data 609
```

```
609  [ 4 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 621
                                                                  snd_ack  607
523  [ 2 , 1 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,E  -1 , -1 ] snd_data 541
                                                                  rcv_data 542
542  [ 2 , 2 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 555
                                                                  snd_ack  556
556  [ 2 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 572
                                                                  snd_data 573
572  [ 0 , 0 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 584
584  [ 1 , 0 , 4 , 4 , 3 , 3 , E  -1 , E  -1 , D2  3 ,E  -1 , -1 ] snd_data 599
                                                                  rcv_data 600
599  [ 2 , 0 , 0 , 1 , 3 , 3 , E  -1 , E  -1 , D2  3 ,D3  4 , -1 ] snd_data 610
                                                                  rcv_data 611
610  [ 3 , 0 , 1 , 2 , 3 , 3 , D0  0 , E  -1 , D2  3 ,D3  4 , -1 ] snd_data 622
                                                                  rcv_data 623
622  [ 4 , 0 , 2 , 3 , 3 , 3 , D0  0 , D1  1 , D2  3 ,D3  4 , -1 ] rcv_data 634
634  [ 4 , 1 , 2 , 3 , 4 , 4 , D0  0 , D1  1 , E  -1 ,D3  4 , -1 ] rcv_data 649
649  [ 4 , 2 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 , -1 ] rcv_data 663
663  [ 4 , 3 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 , -1 ] rcv_data 678
678  [ 4 , 4 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  688
688  [ 4 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0  20
623  [ 3 , 1 , 1 , 2 , 4 , 4 , D0  0 , E  -1 , E  -1 ,D3  4 , -1 ] snd_data 634
                                                                  rcv_data 635
635  [ 3 , 2 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 649
                                                                  rcv_data 650
650  [ 3 , 3 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 663
                                                                  snd_ack  664
664  [ 3 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0  10
                                                                  snd_data 679
679  [ 4 , 0 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 ,  1 ] rcv_ack1  15
                                                                  rcv_data 689
689  [ 4 , 1 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1  23
                                                                  snd_ack  688
611  [ 2 , 1 , 0 , 1 , 4 , 4 , E  -1 , E  -1 , E  -1 ,D3  4 , -1 ] snd_data 623
                                                                  rcv_data 624
624  [ 2 , 2 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 635
                                                                  snd_ack  636
636  [ 2 , 0 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0   0
                                                                  snd_data 651
651  [ 3 , 0 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1   1
                                                                  snd_data 665
                                                                  rcv_data 666
665  [ 4 , 0 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   2
                                                                  rcv_data 680
680  [ 4 , 1 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   5
                                                                  rcv_data 690
690  [ 4 , 2 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2   9
                                                                  snd_ack  688
666  [ 3 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1   3
                                                                  snd_data 680
                                                                  snd_ack  664
600  [ 1 , 1 , 4 , 4 , 4 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 611
                                                                  snd_ack  612
612  [ 1 , 0 , 4 , 4 , 4 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 597
                                                                  snd_data 625
625  [ 2 , 0 , 0 , 1 , 4 , 4 , E  -1 , E  -1 , E  -1 ,D3  4 ,  4 ] rcv_ack1 608
                                                                  snd_data 637
                                                                  rcv_data 638
637  [ 3 , 0 , 1 , 2 , 4 , 4 , D0  0 , E  -1 , E  -1 ,D3  4 ,  4 ] rcv_ack2 620
                                                                  snd_data 652
                                                                  rcv_data 653
652  [ 4 , 0 , 2 , 3 , 4 , 4 , D0  0 , D1  1 , E  -1 ,D3  4 ,  4 ] rcv_ack3 631
                                                                  rcv_data 667
667  [ 4 , 1 , 2 , 3 , 0 , 1 , D0  0 , D1  1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 646
                                                                  rcv_data 681
681  [ 4 , 2 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 659
                                                                  rcv_data 691
691  [ 4 , 3 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 674
                                                                  snd_ack  688
653  [ 3 , 1 , 1 , 2 , 0 , 1 , D0  0 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 632
                                                                  snd_data 667
                                                                  rcv_data 668
668  [ 3 , 2 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 647
                                                                  snd_data 681
                                                                  snd_ack  664
638  [ 2 , 1 , 0 , 1 , 0 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 621
```

```
                                                
```

```
                                                
```

```
                                                
```

```
498  [ 2 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 480
                                                                  snd_data 515
515  [ 3 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 492
                                                                  snd_data 532
                                                                  rcv_data 533
532  [ 4 , 0 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 ,E  -1 ,  1 ] rcv_ack2 507
                                                                  rcv_data 552
552  [ 4 , 1 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,E  -1 ,  1 ] rcv_ack2 523
                                                                  rcv_data 568
568  [ 4 , 2 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2 542
                                                                  snd_ack  566
533  [ 3 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 508
                                                                  snd_data 552
                                                                  snd_ack  531
460  [ 1 , 1 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 471
                                                                  snd_ack  472
472  [ 1 , 0 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 457
                                                                  snd_data 486
486  [ 2 , 0 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 ,  0 ] rcv_ack1 468
                                                                  snd_data 499
                                                                  rcv_data 500
499  [ 3 , 0 , 2 , 2 , 0 , 4 , D0  1 , E  -1 , E  -1 ,D3  0 ,  0 ] rcv_ack2 481
                                                                  snd_data 516
                                                                  rcv_data 517
516  [ 4 , 0 , 3 , 3 , 0 , 4 , D0  1 , D1  2 , E  -1 ,D3  0 ,  0 ] rcv_ack3 493
                                                                  rcv_data 534
534  [ 4 , 1 , 3 , 3 , 1 , 1 , D0  1 , D1  2 , E  -1 ,E  -1 ,  0 ] rcv_ack3 510
                                                                  rcv_data 553
553  [ 4 , 2 , 3 , 3 , 2 , 2 , E  -1 , D1  2 , E  -1 ,E  -1 ,  0 ] rcv_ack3 526
                                                                  rcv_data 569
569  [ 4 , 3 , 3 , 3 , 3 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack3 546
                                                                  snd_ack  566
517  [ 3 , 1 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 494
                                                                  snd_data 534
                                                                  rcv_data 535
535  [ 3 , 2 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 511
                                                                  snd_data 553
                                                                  snd_ack  531
500  [ 2 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1 482
                                                                  snd_data 517
                                                                  snd_ack  498
433  [ 3 , 0 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  4 ] rcv_ack1 444
                                                                  snd_data 445
                                                                  rcv_data 446
445  [ 4 , 0 , 1 , 1 , 4 , 3 , E  -1 , E  -1 , D2  4 ,D3  0 ,  4 ] rcv_ack2 459
                                                                  rcv_data 461
461  [ 4 , 1 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 ,  4 ] rcv_ack2 471
                                                                  rcv_data 473
473  [ 4 , 2 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 485
                                                                  snd_ack  467
446  [ 3 , 1 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 460
                                                                  snd_data 461
                                                                  snd_ack  443
368  [ 1 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 383
                                                                  snd_ack  384
384  [ 1 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 403
                                                                  snd_data 404
403  [ 0 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 417
417  [ 1 , 0 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 , -1 ] snd_data 434
                                                                  rcv_data 435
434  [ 2 , 0 , 0 , 4 , 3 , 2 , E  -1 , D1  3 , D2  4 ,E  -1 , -1 ] snd_data 447
                                                                  rcv_data 448
447  [ 3 , 0 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  4 ,D3  0 , -1 ] snd_data 462
                                                                  rcv_data 463
462  [ 4 , 0 , 2 , 2 , 3 , 2 , D0  1 , D1  3 , D2  4 ,D3  0 , -1 ] rcv_data 474
474  [ 4 , 1 , 2 , 2 , 4 , 3 , D0  1 , E  -1 , D2  4 ,D3  0 , -1 ] rcv_data 487
487  [ 4 , 2 , 2 , 2 , 0 , 4 , D0  1 , E  -1 , E  -1 ,D3  0 , -1 ] rcv_data 501
501  [ 4 , 3 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 , -1 ] rcv_data 518
518  [ 4 , 4 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  536
536  [ 4 , 0 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 543
463  [ 3 , 1 , 1 , 1 , 4 , 3 , E  -1 , E  -1 , D2  4 ,D3  0 , -1 ] snd_data 474
                                                                  rcv_data 475
475  [ 3 , 2 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 , -1 ] snd_data 487
                                                                  rcv_data 488
488  [ 3 , 3 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 501
                                                                  snd_ack  502
```

```
502  [ 3 , 0 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 480
                                                                  snd_data 519
519  [ 4 , 0 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 492
                                                                  rcv_data 537
537  [ 4 , 1 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 508
                                                                  snd_ack  536
448  [ 2 , 1 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 , -1 ] snd_data 463
                                                                  rcv_data 464
464  [ 2 , 2 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 475
                                                                  snd_ack  476
476  [ 2 , 0 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 457
                                                                  snd_data 489
489  [ 3 , 0 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 ,  0 ] rcv_ack1 468
                                                                  snd_data 503
                                                                  rcv_data 504
503  [ 4 , 0 , 2 , 2 , 0 , 4 , D0  1 , E  -1 , E  -1 ,D3  0 ,  0 ] rcv_ack2 481
                                                                  rcv_data 520
520  [ 4 , 1 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 494
                                                                  rcv_data 538
538  [ 4 , 2 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 511
                                                                  snd_ack  536
504  [ 3 , 1 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1 482
                                                                  snd_data 520
                                                                  snd_ack  502
435  [ 1 , 1 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 448
                                                                  snd_ack  449
449  [ 1 , 0 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 432
                                                                  snd_data 465
465  [ 2 , 0 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  4 ] rcv_ack1 444
                                                                  snd_data 477
                                                                  rcv_data 478
477  [ 3 , 0 , 1 , 1 , 4 , 3 , E  -1 , E  -1 , D2  4 ,D3  0 ,  4 ] rcv_ack2 459
                                                                  snd_data 490
                                                                  rcv_data 491
490  [ 4 , 0 , 2 , 2 , 4 , 3 , D0  1 , E  -1 , D2  4 ,D3  0 ,  4 ] rcv_ack3 470
                                                                  rcv_data 505
505  [ 4 , 1 , 2 , 2 , 0 , 4 , D0  1 , E  -1 , E  -1 ,D3  0 ,  4 ] rcv_ack3 484
                                                                  rcv_data 521
521  [ 4 , 2 , 2 , 2 , 1 , 1 , D0  1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 497
                                                                  rcv_data 539
539  [ 4 , 3 , 2 , 2 , 2 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 514
                                                                  snd_ack  536
491  [ 3 , 1 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 ,  4 ] rcv_ack2 471
                                                                  snd_data 505
                                                                  rcv_data 506
506  [ 3 , 2 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 485
                                                                  snd_data 521
                                                                  snd_ack  502
478  [ 2 , 1 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 460
                                                                  snd_data 491
                                                                  snd_ack  476
404  [ 2 , 0 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  3 ] rcv_ack1 417
                                                                  snd_data 418
                                                                  rcv_data 419
418  [ 3 , 0 , 0 , 4 , 3 , 2 , E  -1 , D1  3 , D2  4 ,E  -1 ,  3 ] rcv_ack2 434
                                                                  snd_data 436
                                                                  rcv_data 437
436  [ 4 , 0 , 1 , 1 , 3 , 2 , E  -1 , D1  3 , D2  4 ,D3  0 ,  3 ] rcv_ack3 447
                                                                  rcv_data 450
450  [ 4 , 1 , 1 , 1 , 4 , 3 , E  -1 , E  -1 , D2  4 ,D3  0 ,  3 ] rcv_ack3 463
                                                                  rcv_data 466
466  [ 4 , 2 , 1 , 1 , 0 , 4 , E  -1 , E  -1 , E  -1 ,D3  0 ,  3 ] rcv_ack3 475
                                                                  rcv_data 479
479  [ 4 , 3 , 1 , 1 , 1 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack3 488
                                                                  snd_ack  467
437  [ 3 , 1 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  3 ] rcv_ack2 448
                                                                  snd_data 450
                                                                  rcv_data 451
451  [ 3 , 2 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2 464
                                                                  snd_data 466
                                                                  snd_ack  443
419  [ 2 , 1 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 435
                                                                  snd_data 437
                                                                  snd_ack  416
261  [ 3 , 1 , 1 , 4 , 4 , 2 , E  -1 , D1  4 , D2  0 ,E  -1 , -1 ] snd_data 274
                                                                  rcv_data 275
```

```
275   [ 3 , 2 , 1 , 4 , 0 , 3 , E  -1 , E  -1 , D2  0 ,E  -1 , -1 ] snd_data 290
                                                                    rcv_data 291
291   [ 3 , 3 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 302
                                                                    snd_ack  303
303   [ 3 , 0 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 317
                                                                    snd_data 318
317   [ 0 , 0 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 328
328   [ 1 , 0 , 2 , 1 , 1 , 4 , E  -1 , E  -1 , E  -1 ,D3  1 , -1 ] snd_data 341
                                                                    rcv_data 342
341   [ 2 , 0 , 3 , 2 , 1 , 4 , D0  2 , E  -1 , E  -1 ,D3  1 , -1 ] snd_data 353
                                                                    rcv_data 354
353   [ 3 , 0 , 4 , 3 , 1 , 4 , D0  2 , D1  3 , E  -1 ,D3  1 , -1 ] snd_data 369
                                                                    rcv_data 370
369   [ 4 , 0 , 0 , 4 , 1 , 4 , D0  2 , D1  3 , D2  4 ,D3  1 , -1 ] rcv_data 385
385   [ 4 , 1 , 0 , 4 , 2 , 1 , D0  2 , D1  3 , D2  4 ,E  -1 , -1 ] rcv_data 405
405   [ 4 , 2 , 0 , 4 , 3 , 2 , E  -1 , D1  3 , D2  4 ,E  -1 , -1 ] rcv_data 420
420   [ 4 , 3 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 , -1 ] rcv_data 438
438   [ 4 , 4 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  452
452   [ 4 , 0 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 457
370   [ 3 , 1 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 , -1 ] snd_data 385
                                                                    rcv_data 386
386   [ 3 , 2 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 , -1 ] snd_data 405
                                                                    rcv_data 406
406   [ 3 , 3 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 420
                                                                    snd_ack  421
421   [ 3 , 0 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 432
                                                                    snd_data 439
439   [ 4 , 0 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  4 ] rcv_ack1 444
                                                                    rcv_data 453
453   [ 4 , 1 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 460
                                                                    snd_ack  452
354   [ 2 , 1 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 370
                                                                    rcv_data 371
371   [ 2 , 2 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 386
                                                                    snd_ack  387
387   [ 2 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 403
                                                                    snd_data 407
407   [ 3 , 0 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  3 ] rcv_ack1 417
                                                                    snd_data 422
                                                                    rcv_data 423
422   [ 4 , 0 , 0 , 4 , 3 , 2 , E  -1 , D1  3 , D2  4 ,E  -1 ,  3 ] rcv_ack2 434
                                                                    rcv_data 440
440   [ 4 , 1 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  3 ] rcv_ack2 448
                                                                    rcv_data 454
454   [ 4 , 2 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2 464
                                                                    snd_ack  452
423   [ 3 , 1 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 435
                                                                    snd_data 440
                                                                    snd_ack  421
342   [ 1 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 354
                                                                    snd_ack  355
355   [ 1 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 340
                                                                    snd_data 372
372   [ 2 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 352
                                                                    snd_data 388
                                                                    rcv_data 389
388   [ 3 , 0 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 ,  2 ] rcv_ack2 367
                                                                    snd_data 408
                                                                    rcv_data 409
408   [ 4 , 0 , 0 , 4 , 2 , 1 , D0  2 , D1  3 , D2  4 ,E  -1 ,  2 ] rcv_ack3 382
                                                                    rcv_data 424
424   [ 4 , 1 , 0 , 4 , 3 , 2 , E  -1 , D1  3 , D2  4 ,E  -1 ,  2 ] rcv_ack3 401
                                                                    rcv_data 441
441   [ 4 , 2 , 0 , 4 , 4 , 3 , E  -1 , E  -1 , D2  4 ,E  -1 ,  2 ] rcv_ack3 415
                                                                    rcv_data 455
455   [ 4 , 3 , 0 , 4 , 0 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack3 431
                                                                    snd_ack  452
409   [ 3 , 1 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  2 ] rcv_ack2 383
                                                                    snd_data 424
                                                                    rcv_data 425
425   [ 3 , 2 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 402
                                                                    snd_data 441
                                                                    snd_ack  421
389   [ 2 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 368
                                                                    snd_data 409
                                                                    snd_ack  387
```

```
318   [ 4 , 0 , 2 , 1 , 1 , 4 , E  -1 , E  -1 , E  -1 ,D3  1 ,  1 ] rcv_ack1 328
                                                                    rcv_data 329
329   [ 4 , 1 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 342
                                                                    snd_ack  327
243   [ 2 , 1 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 , -1 ] snd_data 261
                                                                    rcv_data 262
262   [ 2 , 2 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 275
                                                                    snd_ack  276
276   [ 2 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 292
                                                                    snd_data 293
292   [ 0 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 304
304   [ 1 , 0 , 1 , 4 , 0 , 3 , E  -1 , E  -1 , D2  0 ,E  -1 , -1 ] snd_data 319
                                                                    rcv_data 320
319   [ 2 , 0 , 2 , 1 , 0 , 3 , E  -1 , E  -1 , D2  0 ,D3  1 , -1 ] snd_data 330
                                                                    rcv_data 331
330   [ 3 , 0 , 3 , 2 , 0 , 3 , D0  2 , E  -1 , D2  0 ,D3  1 , -1 ] snd_data 343
                                                                    rcv_data 344
343   [ 4 , 0 , 4 , 3 , 0 , 3 , D0  2 , D1  3 , D2  0 ,D3  1 , -1 ] rcv_data 356
356   [ 4 , 1 , 4 , 3 , 1 , 4 , D0  2 , D1  3 , E  -1 ,D3  1 , -1 ] rcv_data 373
373   [ 4 , 2 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 , -1 ] rcv_data 390
390   [ 4 , 3 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 , -1 ] rcv_data 410
410   [ 4 , 4 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  426
426   [ 4 , 0 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 432
344   [ 3 , 1 , 3 , 2 , 1 , 4 , D0  2 , E  -1 , E  -1 ,D3  1 , -1 ] snd_data 356
                                                                    rcv_data 357
357   [ 3 , 2 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 373
                                                                    rcv_data 374
374   [ 3 , 3 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 390
                                                                    snd_ack  391
391   [ 3 , 0 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 403
                                                                    snd_data 411
411   [ 4 , 0 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  3 ] rcv_ack1 417
                                                                    rcv_data 427
427   [ 4 , 1 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 435
                                                                    snd_ack  426
331   [ 2 , 1 , 2 , 1 , 1 , 4 , E  -1 , E  -1 , E  -1 ,D3  1 , -1 ] snd_data 344
                                                                    rcv_data 345
345   [ 2 , 2 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 357
                                                                    snd_ack  358
358   [ 2 , 0 , 2 , 1 , 2 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 340
                                                                    snd_data 375
375   [ 3 , 0 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 352
                                                                    snd_data 392
                                                                    rcv_data 393
392   [ 4 , 0 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 ,  2 ] rcv_ack2 367
                                                                    rcv_data 412
412   [ 4 , 1 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  2 ] rcv_ack2 383
                                                                    rcv_data 428
428   [ 4 , 2 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 402
                                                                    snd_ack  426
393   [ 3 , 1 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 368
                                                                    snd_data 412
                                                                    snd_ack  391
320   [ 1 , 1 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 331
                                                                    snd_ack  332
332   [ 1 , 0 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 317
                                                                    snd_data 346
346   [ 2 , 0 , 2 , 1 , 1 , 4 , E  -1 , E  -1 , E  -1 ,D3  1 ,  1 ] rcv_ack1 328
                                                                    snd_data 359
                                                                    rcv_data 360
359   [ 3 , 0 , 3 , 2 , 1 , 4 , D0  2 , E  -1 , E  -1 ,D3  1 ,  1 ] rcv_ack2 341
                                                                    snd_data 376
                                                                    rcv_data 377
376   [ 4 , 0 , 4 , 3 , 1 , 4 , D0  2 , D1  3 , E  -1 ,D3  1 ,  1 ] rcv_ack3 353
                                                                    rcv_data 394
394   [ 4 , 1 , 4 , 3 , 2 , 1 , D0  2 , D1  3 , E  -1 ,E  -1 ,  1 ] rcv_ack3 370
                                                                    rcv_data 413
413   [ 4 , 2 , 4 , 3 , 3 , 2 , E  -1 , D1  3 , E  -1 ,E  -1 ,  1 ] rcv_ack3 386
                                                                    rcv_data 429
429   [ 4 , 3 , 4 , 3 , 4 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack3 406
                                                                    snd_ack  426
377   [ 3 , 1 , 3 , 2 , 2 , 1 , D0  2 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2 354
                                                                    snd_data 394
                                                                    rcv_data 395
395   [ 3 , 2 , 3 , 2 , 3 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2 371
                                                                    snd_data 413
```

```
                                                
```

```
                                                
```

```
267   [ 3 , 0 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 ,  4 ] rcv_ack1 277
                                                                    snd_data 282
                                                                    rcv_data 283
282   [ 4 , 0 , 1 , 4 , 4 , 2 , E  -1 , D1  4 , D2  0 ,E  -1 ,  4 ] rcv_ack2 294
                                                                    rcv_data 300
300   [ 4 , 1 , 1 , 4 , 0 , 3 , E  -1 , E  -1 , D2  0 ,E  -1 ,  4 ] rcv_ack2 308
                                                                    rcv_data 314
314   [ 4 , 2 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 324
                                                                    snd_ack  312
283   [ 3 , 1 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 295
                                                                    snd_data 300
                                                                    snd_ack  281
202   [ 1 , 1 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 214
                                                                    snd_ack  215
215   [ 1 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 200
                                                                    snd_data 232
232   [ 2 , 0 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 212
                                                                    snd_data 248
                                                                    rcv_data 249
248   [ 3 , 0 , 0 , 3 , 3 , 1 , D0  3 , D1  4 , E  -1 ,E  -1 ,  3 ] rcv_ack2 227
                                                                    snd_data 268
                                                                    rcv_data 269
268   [ 4 , 0 , 1 , 4 , 3 , 1 , D0  3 , D1  4 , D2  0 ,E  -1 ,  3 ] rcv_ack3 242
                                                                    rcv_data 284
284   [ 4 , 1 , 1 , 4 , 4 , 2 , E  -1 , D1  4 , D2  0 ,E  -1 ,  3 ] rcv_ack3 261
                                                                    rcv_data 301
301   [ 4 , 2 , 1 , 4 , 0 , 3 , E  -1 , E  -1 , D2  0 ,E  -1 ,  3 ] rcv_ack3 275
                                                                    rcv_data 315
315   [ 4 , 3 , 1 , 4 , 1 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack3 291
                                                                    snd_ack  312
269   [ 3 , 1 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 ,  3 ] rcv_ack2 243
                                                                    snd_data 284
                                                                    rcv_data 285
285   [ 3 , 2 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2 262
                                                                    snd_data 301
                                                                    snd_ack  281
249   [ 2 , 1 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 228
                                                                    snd_data 269
                                                                    snd_ack  247
178   [ 4 , 0 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  2 ] rcv_ack1 188
                                                                    rcv_data 189
189   [ 4 , 1 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 202
                                                                    snd_ack  187
103   [ 2 , 1 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 , -1 ] snd_data 121
                                                                    rcv_data 122
122   [ 2 , 2 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 135
                                                                    snd_ack  136
136   [ 2 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 152
                                                                    snd_data 153
152   [ 0 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 164
164   [ 1 , 0 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 , -1 ] snd_data 179
                                                                    rcv_data 180
179   [ 2 , 0 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 , -1 ] snd_data 190
                                                                    rcv_data 191
190   [ 3 , 0 , 4 , 2 , 1 , 3 , D0  3 , E  -1 , D2  1 ,D3  2 , -1 ] snd_data 203
                                                                    rcv_data 204
203   [ 4 , 0 , 0 , 3 , 1 , 3 , D0  3 , D1  4 , D2  1 ,D3  2 , -1 ] rcv_data 216
216   [ 4 , 1 , 0 , 3 , 2 , 4 , D0  3 , D1  4 , E  -1 ,D3  2 , -1 ] rcv_data 233
233   [ 4 , 2 , 0 , 3 , 3 , 1 , D0  3 , D1  4 , E  -1 ,E  -1 , -1 ] rcv_data 250
250   [ 4 , 3 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 , -1 ] rcv_data 270
270   [ 4 , 4 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  286
286   [ 4 , 0 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 292
204   [ 3 , 1 , 4 , 2 , 2 , 4 , D0  3 , E  -1 , E  -1 ,D3  2 , -1 ] snd_data 216
                                                                    rcv_data 217
217   [ 3 , 2 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 233
                                                                    rcv_data 234
234   [ 3 , 3 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 250
                                                                    snd_ack  251
251   [ 3 , 0 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 263
                                                                    snd_data 271
271   [ 4 , 0 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 ,  4 ] rcv_ack1 277
                                                                    rcv_data 287
287   [ 4 , 1 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1 295
                                                                    snd_ack  286
191   [ 2 , 1 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 , -1 ] snd_data 204
                                                                    rcv_data 205
```

```
205   [ 2 , 2 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 217
                                                                    snd_ack  218
218   [ 2 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 200
                                                                    snd_data 235
235   [ 3 , 0 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 212
                                                                    snd_data 252
                                                                    rcv_data 253
252   [ 4 , 0 , 0 , 3 , 3 , 1 , D0  3 , D1  4 , E  -1 ,E  -1 ,  3 ] rcv_ack2 227
                                                                    rcv_data 272
272   [ 4 , 1 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 ,  3 ] rcv_ack2 243
                                                                    rcv_data 288
288   [ 4 , 2 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2 262
                                                                    snd_ack  286
253   [ 3 , 1 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 228
                                                                    snd_data 272
                                                                    snd_ack  251
180   [ 1 , 1 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 191
                                                                    snd_ack  192
192   [ 1 , 0 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 177
                                                                    snd_data 206
206   [ 2 , 0 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  2 ] rcv_ack1 188
                                                                    snd_data 219
                                                                    rcv_data 220
219   [ 3 , 0 , 4 , 2 , 2 , 4 , D0  3 , E  -1 , E  -1 ,D3  2 ,  2 ] rcv_ack2 201
                                                                    snd_data 236
                                                                    rcv_data 237
236   [ 4 , 0 , 0 , 3 , 2 , 4 , D0  3 , D1  4 , E  -1 ,D3  2 ,  2 ] rcv_ack3 213
                                                                    rcv_data 254
254   [ 4 , 1 , 0 , 3 , 3 , 1 , D0  3 , D1  4 , E  -1 ,E  -1 ,  2 ] rcv_ack3 230
                                                                    rcv_data 273
273   [ 4 , 2 , 0 , 3 , 4 , 2 , E  -1 , D1  4 , E  -1 ,E  -1 ,  2 ] rcv_ack3 246
                                                                    rcv_data 289
289   [ 4 , 3 , 0 , 3 , 0 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack3 266
                                                                    snd_ack  286
237   [ 3 , 1 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 214
                                                                    snd_data 254
                                                                    rcv_data 255
255   [ 3 , 2 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 231
                                                                    snd_data 273
                                                                    snd_ack  251
220   [ 2 , 1 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 202
                                                                    snd_data 237
                                                                    snd_ack  218
153   [ 3 , 0 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  1 ] rcv_ack1 164
                                                                    snd_data 165
                                                                    rcv_data 166
165   [ 4 , 0 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 ,  1 ] rcv_ack2 179
                                                                    rcv_data 181
181   [ 4 , 1 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  1 ] rcv_ack2 191
                                                                    rcv_data 193
193   [ 4 , 2 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2 205
                                                                    snd_ack  187
166   [ 3 , 1 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 180
                                                                    snd_data 181
                                                                    snd_ack  163
 88   [ 1 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 103
                                                                    snd_ack  104
104   [ 1 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 123
                                                                    snd_data 124
123   [ 0 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 137
137   [ 1 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 , -1 ] snd_data 154
                                                                    rcv_data 155
154   [ 2 , 0 , 2 , 4 , 0 , 2 , E  -1 , D1  0 , D2  1 ,E  -1 , -1 ] snd_data 167
                                                                    rcv_data 168
167   [ 3 , 0 , 3 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 ,D3  2 , -1 ] snd_data 182
                                                                    rcv_data 183
182   [ 4 , 0 , 4 , 2 , 0 , 2 , D0  3 , D1  0 , D2  1 ,D3  2 , -1 ] rcv_data 194
194   [ 4 , 1 , 4 , 2 , 1 , 3 , D0  3 , E  -1 , D2  1 ,D3  2 , -1 ] rcv_data 207
207   [ 4 , 2 , 4 , 2 , 2 , 4 , D0  3 , E  -1 , E  -1 ,D3  2 , -1 ] rcv_data 221
221   [ 4 , 3 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 , -1 ] rcv_data 238
238   [ 4 , 4 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  256
256   [ 4 , 0 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0 263
183   [ 3 , 1 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 , -1 ] snd_data 194
                                                                    rcv_data 195
195   [ 3 , 2 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 , -1 ] snd_data 207
                                                                    rcv_data 208
```

```
208  [ 3 , 3 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 221
                                                                   snd_ack  222
222  [ 3 , 0 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0 200
                                                                   snd_data 239
239  [ 4 , 0 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 212
                                                                   rcv_data 257
257  [ 4 , 1 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1 228
                                                                   snd_ack  256
168  [ 2 , 1 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 , -1 ] snd_data 183
                                                                   rcv_data 184
184  [ 2 , 2 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 195
                                                                   snd_ack  196
196  [ 2 , 0 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 177
                                                                   snd_data 209
209  [ 3 , 0 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  2 ] rcv_ack1 188
                                                                   snd_data 223
                                                                   rcv_data 224
223  [ 4 , 0 , 4 , 2 , 2 , 4 , D0  3 , E  -1 , E  -1 ,D3  2 ,  2 ] rcv_ack2 201
                                                                   rcv_data 240
240  [ 4 , 1 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 214
                                                                   rcv_data 258
258  [ 4 , 2 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2 231
                                                                   snd_ack  256
224  [ 3 , 1 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1 202
                                                                   snd_data 240
                                                                   snd_ack  222
155  [ 1 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 168
                                                                   snd_ack  169
169  [ 1 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 152
                                                                   snd_data 185
185  [ 2 , 0 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  1 ] rcv_ack1 164
                                                                   snd_data 197
                                                                   rcv_data 198
197  [ 3 , 0 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 ,  1 ] rcv_ack2 179
                                                                   snd_data 210
                                                                   rcv_data 211
210  [ 4 , 0 , 4 , 2 , 1 , 3 , D0  3 , E  -1 , D2  1 ,D3  2 ,  1 ] rcv_ack3 190
                                                                   rcv_data 225
225  [ 4 , 1 , 4 , 2 , 2 , 4 , D0  3 , E  -1 , E  -1 ,D3  2 ,  1 ] rcv_ack3 204
                                                                   rcv_data 241
241  [ 4 , 2 , 4 , 2 , 3 , 1 , D0  3 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack3 217
                                                                   rcv_data 259
259  [ 4 , 3 , 4 , 2 , 4 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack3 234
                                                                   snd_ack  256
211  [ 3 , 1 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  1 ] rcv_ack2 191
                                                                   snd_data 225
                                                                   rcv_data 226
226  [ 3 , 2 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack2 205
                                                                   snd_data 241
                                                                   snd_ack  222
198  [ 2 , 1 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 180
                                                                   snd_data 211
                                                                   snd_ack  196
124  [ 2 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 ,  0 ] rcv_ack1 137
                                                                   snd_data 138
                                                                   rcv_data 139
138  [ 3 , 0 , 2 , 4 , 0 , 2 , E  -1 , D1  0 , D2  1 ,E  -1 ,  0 ] rcv_ack2 154
                                                                   snd_data 156
                                                                   rcv_data 157
156  [ 4 , 0 , 3 , 1 , 0 , 2 , E  -1 , D1  0 , D2  1 ,D3  2 ,  0 ] rcv_ack3 167
                                                                   rcv_data 170
170  [ 4 , 1 , 3 , 1 , 1 , 3 , E  -1 , E  -1 , D2  1 ,D3  2 ,  0 ] rcv_ack3 183
                                                                   rcv_data 186
186  [ 4 , 2 , 3 , 1 , 2 , 4 , E  -1 , E  -1 , E  -1 ,D3  2 ,  0 ] rcv_ack3 195
                                                                   rcv_data 199
199  [ 4 , 3 , 3 , 1 , 3 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack3 208
                                                                   snd_ack  187
157  [ 3 , 1 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  0 ] rcv_ack2 168
                                                                   snd_data 170
                                                                   rcv_data 171
171  [ 3 , 2 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 184
                                                                   snd_data 186
                                                                   snd_ack  163
139  [ 2 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1 155
                                                                   snd_data 157
                                                                   snd_ack  136
```

```
  8   [ 3 , 1 , 3 , 4 , 1 , 2 , E  -1 , D1  1 , D2  2 ,E  -1 , -1 ] snd_data  12
                                                                    rcv_data  13
 13   [ 3 , 2 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 , -1 ] snd_data  18
                                                                    rcv_data  19
 19   [ 3 , 3 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  26
                                                                    snd_ack   27
 27   [ 3 , 0 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0  37
                                                                    snd_data  38
 37   [ 0 , 0 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  48
 48   [ 1 , 0 , 4 , 1 , 3 , 4 , E  -1 , E  -1 , E  -1 ,D3  3 , -1 ] snd_data  61
                                                                    rcv_data  62
 61   [ 2 , 0 , 0 , 2 , 3 , 4 , D0  4 , E  -1 , E  -1 ,D3  3 , -1 ] snd_data  73
                                                                    rcv_data  74
 73   [ 3 , 0 , 1 , 3 , 3 , 4 , D0  4 , D1  0 , E  -1 ,D3  3 , -1 ] snd_data  89
                                                                    rcv_data  90
 89   [ 4 , 0 , 2 , 4 , 3 , 4 , D0  4 , D1  0 , D2  1 ,D3  3 , -1 ] rcv_data 105
105   [ 4 , 1 , 2 , 4 , 4 , 1 , D0  4 , D1  0 , D2  1 ,E  -1 , -1 ] rcv_data 125
125   [ 4 , 2 , 2 , 4 , 0 , 2 , E  -1 , D1  0 , D2  1 ,E  -1 , -1 ] rcv_data 140
140   [ 4 , 3 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 , -1 ] rcv_data 158
158   [ 4 , 4 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  172
172   [ 4 , 0 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0 177
 90   [ 3 , 1 , 1 , 3 , 4 , 1 , D0  4 , D1  0 , E  -1 ,E  -1 , -1 ] snd_data 105
                                                                    rcv_data 106
106   [ 3 , 2 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 , -1 ] snd_data 125
                                                                    rcv_data 126
126   [ 3 , 3 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 140
                                                                    snd_ack  141
141   [ 3 , 0 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0 152
                                                                    snd_data 159
159   [ 4 , 0 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  1 ] rcv_ack1 164
                                                                    rcv_data 173
173   [ 4 , 1 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack1 180
                                                                    snd_ack  172
 74   [ 2 , 1 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  90
                                                                    rcv_data  91
 91   [ 2 , 2 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data 106
                                                                    snd_ack  107
107   [ 2 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 123
                                                                    snd_data 127
127   [ 3 , 0 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 ,  0 ] rcv_ack1 137
                                                                    snd_data 142
                                                                    rcv_data 143
142   [ 4 , 0 , 2 , 4 , 0 , 2 , E  -1 , D1  0 , D2  1 ,E  -1 ,  0 ] rcv_ack2 154
                                                                    rcv_data 160
160   [ 4 , 1 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  0 ] rcv_ack2 168
                                                                    rcv_data 174
174   [ 4 , 2 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack2 184
                                                                    snd_ack  172
143   [ 3 , 1 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack1 155
                                                                    snd_data 160
                                                                    snd_ack  141
 62   [ 1 , 1 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  74
                                                                    snd_ack   75
 75   [ 1 , 0 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0  60
                                                                    snd_data  92
 92   [ 2 , 0 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1  72
                                                                    snd_data 108
                                                                    rcv_data 109
108   [ 3 , 0 , 1 , 3 , 4 , 1 , D0  4 , D1  0 , E  -1 ,E  -1 ,  4 ] rcv_ack2  87
                                                                    snd_data 128
                                                                    rcv_data 129
128   [ 4 , 0 , 2 , 4 , 4 , 1 , D0  4 , D1  0 , D2  1 ,E  -1 ,  4 ] rcv_ack3 102
                                                                    rcv_data 144
144   [ 4 , 1 , 2 , 4 , 0 , 2 , E  -1 , D1  0 , D2  1 ,E  -1 ,  4 ] rcv_ack3 121
                                                                    rcv_data 161
161   [ 4 , 2 , 2 , 4 , 1 , 3 , E  -1 , E  -1 , D2  1 ,E  -1 ,  4 ] rcv_ack3 135
                                                                    rcv_data 175
175   [ 4 , 3 , 2 , 4 , 2 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack3 151
                                                                    snd_ack  172
129   [ 3 , 1 , 1 , 3 , 0 , 2 , E  -1 , D1  0 , E  -1 ,E  -1 ,  4 ] rcv_ack2 103
                                                                    snd_data 144
                                                                    rcv_data 145
145   [ 3 , 2 , 1 , 3 , 1 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack2 122
                                                                    snd_data 161
                                                                    snd_ack  141
109   [ 2 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1  88
```

```
                                                
```

```
115   [ 3 , 2 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2  91
                                                                    snd_data 133
                                                                    snd_ack  111
 80   [ 2 , 1 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1  62
                                                                    snd_data  97
                                                                    snd_ack   78
 21   [ 3 , 0 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 ,  2 ] rcv_ack1  28
                                                                    snd_data  29
                                                                    rcv_data  30
 29   [ 4 , 0 , 4 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 ,D3  3 ,  2 ] rcv_ack2  39
                                                                    rcv_data  41
 41   [ 4 , 1 , 4 , 1 , 3 , 4 , E  -1 , E  -1 , E  -1 ,D3  3 ,  2 ] rcv_ack2  51
                                                                    rcv_data  53
 53   [ 4 , 2 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack2  65
                                                                    snd_ack   47
 30   [ 3 , 1 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack1  40
                                                                    snd_data  41
                                                                    snd_ack   27
  3   [ 1 , 1 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data   5
                                                                    snd_ack    6
  6   [ 1 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  1 ] rcv_ack0  10
                                                                    snd_data  11
 10   [ 0 , 0 , 1 , 2 , 1 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  15
 15   [ 1 , 0 , 2 , 3 , 1 , 2 , E  -1 , D1  1 , E  -1 ,E  -1 , -1 ] snd_data  22
                                                                    rcv_data  23
 22   [ 2 , 0 , 3 , 4 , 1 , 2 , E  -1 , D1  1 , D2  2 ,E  -1 , -1 ] snd_data  31
                                                                    rcv_data  32
 31   [ 3 , 0 , 4 , 1 , 1 , 2 , E  -1 , D1  1 , D2  2 ,D3  3 , -1 ] snd_data  42
                                                                    rcv_data  43
 42   [ 4 , 0 , 0 , 2 , 1 , 2 , D0  4 , D1  1 , D2  2 ,D3  3 , -1 ] rcv_data  54
 54   [ 4 , 1 , 0 , 2 , 2 , 3 , D0  4 , E  -1 , D2  2 ,D3  3 , -1 ] rcv_data  67
 67   [ 4 , 2 , 0 , 2 , 3 , 4 , D0  4 , E  -1 , E  -1 ,D3  3 , -1 ] rcv_data  81
 81   [ 4 , 3 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 , -1 ] rcv_data  98
 98   [ 4 , 4 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_ack  116
116   [ 4 , 0 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  0 ] rcv_ack0 123
 43   [ 3 , 1 , 4 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 ,D3  3 , -1 ] snd_data  54
                                                                    rcv_data  55
 55   [ 3 , 2 , 4 , 1 , 3 , 4 , E  -1 , E  -1 , E  -1 ,D3  3 , -1 ] snd_data  67
                                                                    rcv_data  68
 68   [ 3 , 3 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  81
                                                                    snd_ack   82
 82   [ 3 , 0 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack0  60
                                                                    snd_data  99
 99   [ 4 , 0 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1  72
                                                                    rcv_data 117
117   [ 4 , 1 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  4 ] rcv_ack1  88
                                                                    snd_ack  116
 32   [ 2 , 1 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 , -1 ] snd_data  43
                                                                    rcv_data  44
 44   [ 2 , 2 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  55
                                                                    snd_ack   56
 56   [ 2 , 0 , 3 , 4 , 3 , 4 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack0  37
                                                                    snd_data  69
 69   [ 3 , 0 , 4 , 1 , 3 , 4 , E  -1 , E  -1 , E  -1 ,D3  3 ,  3 ] rcv_ack1  48
                                                                    snd_data  83
                                                                    rcv_data  84
 83   [ 4 , 0 , 0 , 2 , 3 , 4 , D0  4 , E  -1 , E  -1 ,D3  3 ,  3 ] rcv_ack2  61
                                                                    rcv_data 100
100   [ 4 , 1 , 0 , 2 , 4 , 1 , D0  4 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2  74
                                                                    rcv_data 118
118   [ 4 , 2 , 0 , 2 , 0 , 2 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack2  91
                                                                    snd_ack  116
 84   [ 3 , 1 , 4 , 1 , 4 , 1 , E  -1 , E  -1 , E  -1 ,E  -1 ,  3 ] rcv_ack1  62
                                                                    snd_data 100
                                                                    snd_ack   82
 23   [ 1 , 1 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 , -1 ] snd_data  32
                                                                    snd_ack   33
 33   [ 1 , 0 , 2 , 3 , 2 , 3 , E  -1 , E  -1 , E  -1 ,E  -1 ,  2 ] rcv_ack0  20
                                                                    snd_data  45
 45   [ 2 , 0 , 3 , 4 , 2 , 3 , E  -1 , E  -1 , D2  2 ,E  -1 ,  2 ] rcv_ack1  28
                                                                    snd_data  57
                                                                    rcv_data  58
 57   [ 3 , 0 , 4 , 1 , 2 , 3 , E  -1 , E  -1 , D2  2 ,D3  3 ,  2 ] rcv_ack2  39
                                                                    snd_data  70
                                                                    rcv_data  71
 70   [ 4 , 0 , 0 , 2 , 2 , 3 , D0  4 , E  -1 , D2  2 ,D3  3 ,  2 ] rcv_ack3  50
```

```
                                                
```

System State GRAPH

0	[0 , 0]	snd_data	[1 , 0]	1
1	[1 , 0]	snd_data	[2 , 0]	2
		rcv_data	[1 , 1]	3
2	[2 , 0]	snd_data	[3 , 0]	4
		rcv_data	[2 , 1]	5
4	[3 , 0]	snd_data	[4 , 0]	7
		rcv_data	[3 , 1]	8
7	[4 , 0]	rcv_data	[4 , 1]	11
11	[4 , 1]	rcv_data	[4 , 2]	16
16	[4 , 2]	rcv_data	[4 , 3]	21
21	[4 , 3]	rcv_data	[4 , 4]	27
27	[4 , 4]	snd_ack	[4 , 0]	31
31	[4 , 0]	rcv_ack0	[0 , 0]	0
8	[3 , 1]	snd_data	[4 , 1]	11
		rcv_data	[3 , 2]	12
12	[3 , 2]	snd_data	[4 , 2]	16
		rcv_data	[3 , 3]	17
17	[3 , 3]	snd_data	[4 , 3]	21
		snd_ack	[3 , 0]	22
22	[3 , 0]	rcv_ack0	[0 , 0]	0
		snd_data	[4 , 0]	28
28	[4 , 0]	rcv_ack1	[1 , 0]	1
		rcv_data	[4 , 1]	32
32	[4 , 1]	rcv_ack1	[1 , 1]	3
		snd_ack	[4 , 0]	31
5	[2 , 1]	snd_data	[3 , 1]	8
		rcv_data	[2 , 2]	9
9	[2 , 2]	snd_data	[3 , 2]	12
		snd_ack	[2 , 0]	13
13	[2 , 0]	rcv_ack0	[0 , 0]	0
		snd_data	[3 , 0]	18
18	[3 , 0]	rcv_ack1	[1 , 0]	1
		snd_data	[4 , 0]	23
		rcv_data	[3 , 1]	24
23	[4 , 0]	rcv_ack2	[2 , 0]	2
		rcv_data	[4 , 1]	29
29	[4 , 1]	rcv_ack2	[2 , 1]	5
		rcv_data	[4 , 2]	33
33	[4 , 2]	rcv_ack2	[2 , 2]	9
		snd_ack	[4 , 0]	31
24	[3 , 1]	rcv_ack1	[1 , 1]	3
		snd_data	[4 , 1]	29
		snd_ack	[3 , 0]	22
3	[1 , 1]	snd_data	[2 , 1]	5
		snd_ack	[1 , 0]	6
6	[1 , 0]	rcv_ack0	[0 , 0]	0
		snd_data	[2 , 0]	10
10	[2 , 0]	rcv_ack1	[1 , 0]	1
		snd_data	[3 , 0]	14
		rcv_data	[2 , 1]	15
14	[3 , 0]	rcv_ack2	[2 , 0]	2
		snd_data	[4 , 0]	19
		rcv_data	[3 , 1]	20
19	[4 , 0]	rcv_ack3	[3 , 0]	4
		rcv_data	[4 , 1]	25
25	[4 , 1]	rcv_ack3	[3 , 1]	8
		rcv_data	[4 , 2]	30
30	[4 , 2]	rcv_ack3	[3 , 2]	12
		rcv_data	[4 , 3]	34
34	[4 , 3]	rcv_ack3	[3 , 3]	17
		snd_ack	[4 , 0]	31
20	[3 , 1]	rcv_ack2	[2 , 1]	5
		snd_data	[4 , 1]	25
		rcv_data	[3 , 2]	26
26	[3 , 2]	rcv_ack2	[2 , 2]	9
		snd_data	[4 , 2]	30
		snd_ack	[3 , 0]	22
15	[2 , 1]	rcv_ack1	[1 , 1]	3
		snd_data	[3 , 1]	20
		snd_ack	[2 , 0]	13

Machine 1 Array Contents			
From	To	Transition	Executed
0	1	snd_data	yes
1	0	rcv_ack0	yes
1	2	snd_data	yes
2	0	rcv_ack0	yes
2	1	rcv_ack1	yes
2	3	snd_data	yes
3	0	rcv_ack0	yes
3	1	rcv_ack1	yes
3	2	rcv_ack2	yes
3	4	snd_data	yes
4	0	rcv_ack0	yes
4	1	rcv_ack1	yes
4	2	rcv_ack2	yes
4	3	rcv_ack3	yes

Machine 2 Array Contents			
From	To	Transition	Executed
0	1	rcv_data	yes
1	2	rcv_data	yes
1	0	snd_ack	yes
2	3	rcv_data	yes
2	0	snd_ack	yes
3	4	rcv_data	yes
3	0	snd_ack.	yes
4	0	snd_ack	yes

APPENDIX I (SCM) SELECTIVE REPEAT, W=1

INPUT

```
start
machine 1
state 0
trans snd_data1 1
state 1
trans rcv_ack1 2
state 2
trans adv_win1 0
machine 2
state 0
trans rcv_data1 1
state 1
trans snd_ack1 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is

   type scm_transition_type  is (snd_data1, rcv_data1, snd_ack1,
                                 rcv_ack1, adv_win1, unused);
   type buffer_type   is (d1,e,a1);
   type boolean_type    is (t,f);

   type buffer_array_type is array(1..1) of buffer_type;
   type boolean_array_type is array(1..1) of boolean_type;

   type machine1_state_type is
      record
         state_number      : natural              := 0;
         out_buffer        : buffer_array_type    := (others=>d1);
         ack_rec           : boolean_array_type   := (others=>f);
         current           : integer range 1..1   := 1;
   end record;

   type machine2_state_type is
      record
         state_number      : natural              := 0;
         in_buffer         : buffer_array_type    := (others=>e);
         pkt_rec           : boolean_array_type   := (others=>f);
         current           : integer range 1..1   := 1;
   end record;

   type global_variable_type is
      record
         DATA              : buffer_type          := e;
    end record;

end definitions;
```

PREDICATE-ACTION

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if (local.out_buffer(1) /= E) then
          Push(transition_stack,snd_data1);
      end if;
      if ((local.ack_rec(1)=f) and GLOBAL.DATA=A1) then
          Push(transition_stack,rcv_ack1);
      end if;
      Push(transition_stack,adv_win1);
      return transition_stack;
end Analyze_Predicates_Machine1;


separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if ((GLOBAL.DATA = D1) and (local.pkt_rec(1)=f)) then
          Push(transition_stack,rcv_data1);
      end if;
      if (local.pkt_rec(1)=t) then
         Push(transition_stack,snd_ack1);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2; -- this returned value is then checked against the
machine arrays
                                    -- to determine if indeed this transition can be
taken


separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
          when (snd_data1) =>
             out_system_state.GLOBAL_VARIABLES.DATA :=
                 in_system_state.machine1_state.out_buffer(1);
          when (rcv_ack1) =>
              out_system_state.machine1_state.ack_rec(1) := t;
              out_system_state.GLOBAL_VARIABLES.DATA := e;
              out_system_state.machine1_state.current := 1;
          when (rcv_data1) =>
               out_system_state.machine2_state.in_buffer(1) :=
                  in_system_state.GLOBAL_VARIABLES.DATA;
               out_system_state.GLOBAL_VARIABLES.DATA := e;
               out_system_state.machine2_state.pkt_rec(1):= t;
          when (snd_ack1) =>
              out_system_state.GLOBAL_VARIABLES.DATA := a1;
              out_system_state.machine2_state.pkt_rec(1) := f;
              out_system_state.machine2_state.in_buffer(1) := e;
          when (adv_win1)   =>
               out_system_state.ma-
chine1_state.ack_rec(in_system_state.machine1_state.current) := f;
          when others =>
             put_line("There is an error in the Action procedure");
      end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number));
   put(" ," & integer'image(tuple.machine2_state.state_number));
   put(" , ");
   put(tuple.machine1_state.out_buffer(1), width => 1);
   put(" , ");
   put(tuple.machine1_state.ack_rec(1),width=>2);
   put(" , ");
   put(tuple.machine2_state.in_buffer(1), width => 1);
   put(" , ");
   put(tuple.machine2_state.pkt_rec(1),width=>2);
   put(" , ");
   put(tuple.GLOBAL_VARIABLES.DATA, width =>2);
   put(" ]");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number));
   put(reach," ," & integer'image(tuple.machine2_state.state_number));
   put(reach," , ");
   put(reach,tuple.machine1_state.out_buffer(1), width => 1);
   put(reach," , ");
   put(reach,tuple.machine1_state.ack_rec(1),width=>2);
   put(reach," , ");
   put(reach,tuple.machine2_state.in_buffer(1), width => 1);
   put(reach," , ");
   put(reach,tuple.machine2_state.pkt_rec(1),width=>2);
   put(reach," , ");
   put(reach,tuple.GLOBAL_VARIABLES.DATA, width =>2);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;

separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
      Error_flag := true;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

OUTPUT

REACHABILITY ANALYSIS of : sel_rep_w1

Global State GRAPH

```
0   [ 0 , 0 , D1 , F  , E , F  , E  ]        snd_data1   1
1   [ 1 , 0 , D1 , F  , E , F  , D1 ]        rcv_data1   2
2   [ 1 , 1 , D1 , F  , D1 , T  , E  ]       snd_ack1    3
3   [ 1 , 0 , D1 , F  , E , F  , A1 ]        rcv_ack1    4
4   [ 2 , 0 , D1 , T  , E , F  , E  ]        adv_win1    0
```

System State GRAPH

```
0   [ 0 , 0 ]    snd_data1   [ 1 , 0 ]  1
1   [ 1 , 0 ]    rcv_data1   [ 1 , 1 ]  2
2   [ 1 , 1 ]    snd_ack1    [ 1 , 0 ]  3
3   [ 1 , 0 ]    rcv_ack1    [ 2 , 0 ]  4
4   [ 2 , 0 ]    adv_win1    [ 0 , 0 ]  0
```

Machine 1 Array Contents

From	To	Transition	Executed
0	1	snd_data1	yes
1	2	rcv_ack1	yes
2	0	adv_win1	yes

Machine 2 Array Contents

From	To	Transition	Executed
0	1	rcv_data1	yes
1	0	snd_ack1	yes

APPENDIX J (SCM) SELECTIVE REPEAT, W=2

INPUT (FSM)

```
start
machine 1
state 0
trans snd_data1 1
state 1
trans rcv_ack1 2
state 2
trans adv_win1 0
machine 2
state 0
trans rcv_data1 1
state 1
trans snd_ack1 0
initial_state 0 0
finish
```

VARIABLE DEFINITIONS

```
package definitions is

   type scm_transition_type  is (snd_data, rcv_data,
                                 snd_ack,  rcv_ack,
                                 adv_win1, unused);
   type buffer_type   is (d1,d2,e,a1,a2);
   type boolean_type    is (t,f);
   subtype ack_buffer_type is buffer_type range e..a2;
   subtype data_buffer_type is buffer_type range d1..e;

   type ack_array_type is array(1..2) of ack_buffer_type;
   type data_array_type is array(1..2) of data_buffer_type;

   type boolean_array_type is array(1..2) of boolean_type;

   type machine1_state_type is
      record
         state_number      : natural              := 0;
         out_buffer        : data_array_type      := (d1,d2);
         ack_rec           : boolean_array_type   := (others=>f);
         current           : integer range 1..2   := 1;
         hold              : boolean_type         := f;
   end record;

   type machine2_state_type is
      record
         state_number     : natural               := 0;
         in_buffer        : data_array_type       := (others=>e);
         pkt_rec          : boolean_array_type    := (others=>f);
   end record;

   type global_variable_type is
      record
         DATA             : data_array_type      := (others=>e);
         CONTROL          : ack_array_type       := (others=>e);
    end record;

end definitions;
```

```
separate (main)
function  Analyze_Predicates_Machine1(local : machine1_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if (((local.hold = f) and (local.out_buffer(1) /= E) and (GLOBAL.DATA(1)=E)) or
          ((local.hold = f) and (local.out_buffer(2) /= E) and (GLOBAL.DATA(2)=E))) then
          Push(transition_stack,snd_data);
      end if;
      if (((local.ack_rec(1)=f) and (GLOBAL.CONTROL(1) = A1)) or
          ((local.ack_rec(2)=f) and (GLOBAL.CONTROL(2) = A2))) then
          Push(transition_stack,rcv_ack);
      end if;
      Push(transition_stack,adv_win1);
      return transition_stack;
end Analyze_Predicates_Machine1;

separate (main)
function Analyze_Predicates_Machine2(local : machine2_state_type;
                                      GLOBAL: global_variable_type) return transition_-
stack_package.stack is
begin
      MakeEmpty(transition_stack);
      if (((GLOBAL.DATA(1) = D1) and (GLOBAL.DATA(2) = E) and (local.pkt_rec(1)=f)) or
          ((GLOBAL.DATA(1) = E) and (GLOBAL.DATA(2) = D2) and (local.pkt_rec(2)=f)) or
          ((GLOBAL.DATA(1) = D1) and (GLOBAL.DATA(2) = D2) and (local.pkt_rec(1)=t) and
           (local.pkt_rec(2)=f))                                                      or
          ((GLOBAL.DATA(1) = D1) and (GLOBAL.DATA(2) = D2) and (local.pkt_rec(1)=f) and
           (local.pkt_rec(2)=f))) then
          Push(transition_stack,rcv_data);
      end if;
      if (((GLOBAL.CONTROL(1)=E) and (local.pkt_rec(1)=t)) or
          ((GLOBAL.CONTROL(2)=E) and (local.pkt_rec(2)=t))) then
         Push(transition_stack,snd_ack);
      end if;
      return transition_stack;
end Analyze_Predicates_Machine2; -- this returned value is then checked against the
machine arrays
                                 -- to determine if indeed this transition can be
taken

separate (main)
procedure Action(in_system_state  : in out Gstate_record_type;
                 in_transition    : in out scm_transition_type;
                 out_system_state : in out Gstate_record_type) is

      temp : integer := 0;
begin
      case (in_transition) is
          when (snd_data) =>
             out_system_state.GLOBAL_VARIABLES.-
DATA(in_system_state.machine1_state.current) :=
                in_system_state.machine1_state.out_-
buffer(in_system_state.machine1_state.current);
            out_system_state.machine1_state.current := in_system_state.machine1_state.-
current;
            begin
            case (in_system_state.machine1_state.current) is
            when 1 =>
               out_system_state.machine1_state.current := 2;
            when 2 =>
               --out_system_state.machine1_state.current := 1;
               out_system_state.machine1_state.hold := t;
            when others =>
               put_line("error in the action procedure");
            end case;
            end;
          when (rcv_ack) =>
             if(in_system_state.GLOBAL_VARIABLES.CONTROL(1)=A1) then
             out_system_state.machine1_state.ack_rec(1) := t;
                out_system_state.GLOBAL_VARIABLES.CONTROL(1) := e;
             out_system_state.machine1_state.current := 1;
```

```
    else
    if (in_system_state.GLOBAL_VARIABLES.CONTROL(2)=A2) then
          out_system_state.machine1_state.ack_rec(2) := t;
          out_system_state.GLOBAL_VARIABLES.CONTROL(2) := e;
          out_system_state.machine1_state.current := 2;
       end if;
    end if;
 when (rcv_data) =>
  if (in_system_state.GLOBAL_VARIABLES.DATA(1) = D1) then
    out_system_state.machine2_state.in_buffer(1) :=
       in_system_state.GLOBAL_VARIABLES.DATA(1);
     out_system_state.GLOBAL_VARIABLES.DATA(1) := e;
     out_system_state.machine2_state.pkt_rec(1):= t;
  else
    if (in_system_state.GLOBAL_VARIABLES.DATA(2) = D2) then
      out_system_state.machine2_state.in_buffer(2) :=
       in_system_state.GLOBAL_VARIABLES.DATA(2);
      out_system_state.GLOBAL_VARIABLES.DATA(2) := e;
      out_system_state.machine2_state.pkt_rec(2):= t;
    end if;
  end if;
when (snd_ack) =>
   if (in_system_state.machine2_state.pkt_rec(1)=t) then
       out_system_state.GLOBAL_VARIABLES.CONTROL(1) := a1;
    out_system_state.machine2_state.pkt_rec(1) := f;
       out_system_state.machine2_state.in_buffer(1) := e;
   else
   if (in_system_state.machine2_state.pkt_rec(2)=t) then
         out_system_state.GLOBAL_VARIABLES.CONTROL(2) := a2;
         out_system_state.machine2_state.pkt_rec(2) := f;
         out_system_state.machine2_state.in_buffer(2) := e;
       end if;
   end if;
when (adv_win1)   =>
 if ((in_system_state.machine1_state.ack_rec(1)=t) and
     (in_system_state.machine1_state.ack_rec(2)=f) and
     (in_system_state.machine1_state.hold = t) and
     (in_system_state.machine1_state.current = 1) and
     (in_system_state.GLOBAL_VARIABLES.DATA(1) = E) and
     (in_system_state.GLOBAL_VARIABLES.DATA(2) = D2)) then
     out_system_state.machine1_state.ack_rec(1) :=f;
     out_system_state.machine1_state.ack_rec(2) :=f;
     out_system_state.machine1_state.hold :=f;
     out_system_state.machine1_state.current :=2;
     out_system_state.GLOBAL_VARIABLES.DATA(1) := D1;
     out_system_state.GLOBAL_VARIABLES.DATA(2) := E;

  elsif
    ((in_system_state.machine1_state.ack_rec(1)=t) and
     (in_system_state.machine1_state.ack_rec(2)=f) and
     (in_system_state.machine1_state.hold = t) and
     (in_system_state.machine1_state.current = 1) and
     (in_system_state.machine2_state.in_buffer(1) = E) and
     (in_system_state.machine2_state.in_buffer(2) = D2) and
     (in_system_state.machine2_state.pkt_rec(1) = f) and
     (in_system_state.machine2_state.pkt_rec(2) = t)) then
     out_system_state.machine1_state.ack_rec(1) :=f;
     out_system_state.machine1_state.ack_rec(2) :=f;
     out_system_state.machine1_state.hold :=f;
     out_system_state.machine1_state.current :=2;
     out_system_state.machine2_state.in_buffer(1) := D1;
     out_system_state.machine2_state.in_buffer(2) := E;
     out_system_state.machine2_state.pkt_rec(1) := t;
     out_system_state.machine2_state.pkt_rec(2) := f;
  elsif
    ((in_system_state.machine1_state.ack_rec(1)=t) and
     (in_system_state.machine1_state.ack_rec(2)=f) and
     (in_system_state.machine1_state.hold = t) and
     (in_system_state.machine1_state.current = 1) and
     (in_system_state.GLOBAL_VARIABLES.CONTROL(1) = E) and
     (in_system_state.GLOBAL_VARIABLES.CONTROL(2) = A2)) then
     out_system_state.machine1_state.ack_rec(1) :=f;
     out_system_state.machine1_state.ack_rec(2) :=f;
     out_system_state.machine1_state.hold :=f;
     out_system_state.machine1_state.current :=2;
     out_system_state.GLOBAL_VARIABLES.CONTROL(1) := A1;
```

```
            out_system_state.GLOBAL_VARIABLES.CONTROL(2) := E;
            else


            if ((in_system_state.machine1_state.ack_rec(1)=t) and
               (in_system_state.GLOBAL_VARIABLES.DATA(2)/=E)) then
            out_system_state.machine1_state.hold := t;
            elsif ((in_system_state.machine1_state.ack_rec(1)=t) and
                   (in_system_state.machine1_state.ack_rec(2)=f) and
                   (in_system_state.GLOBAL_VARIABLES.DATA(2)=E)) then
                      out_system_state.machine1_state.ack_rec(1) := f;
            elsif ((in_system_state.machine1_state.ack_rec(1)=f) and
                   (in_system_state.machine1_state.ack_rec(2)=t) and
                   (in_system_state.machine1_state.hold = t)) then
                      out_system_state.machine1_state.ack_rec(2) := f;
                      out_system_state.machine1_state.hold := f;
            else
              if ((in_system_state.machine1_state.ack_rec(1)=t) and
                  (in_system_state.machine1_state.ack_rec(2)=t) and
                  (in_system_state.GLOBAL_VARIABLES.DATA(1) =E) and
                  (in_system_state.GLOBAL_VARIABLES.DATA(2)=E)) then
                     out_system_state.machine1_state.hold := f;
                     out_system_state.machine1_state.ack_rec(1) := f;
                     out_system_state.machine1_state.ack_rec(2) := f;
              end if;
            end if;
            out_system_state.machine1_state.current := 1;
            end if;
          when others =>
             put_line("There is an error in the Action procedure");
      end case;
end Action;
```

OUTPUT FORMAT

```
separate (main)
procedure output_Gtuple(tuple : in out Gstate_record_type) is
begin
   put("   [" & integer'image(tuple.machine1_state.state_number));
   put(" ," & integer'image(tuple.machine2_state.state_number));
   put(",");
   put(tuple.machine1_state.out_buffer(1), width => 2);
   put(",");
   put(tuple.machine1_state.out_buffer(2), width => 2);
   put(",");
   put(tuple.machine1_state.ack_rec(1),width=>1);
   put(",");
   put(tuple.machine1_state.ack_rec(2),width=>1);
   put(",");
   put(tuple.machine1_state.hold, width=>1);
   put(",");
   put(tuple.machine2_state.in_buffer(1), width => 2);
   put(",");
   put(tuple.machine2_state.in_buffer(2), width => 2);
   put(",");
   put(tuple.machine2_state.pkt_rec(1),width=>1);
   put(",");
   put(tuple.machine2_state.pkt_rec(2),width=>1);
   put(",");
   put(tuple.GLOBAL_VARIABLES.DATA(1), width =>2);
   put(" ");
   put(tuple.GLOBAL_VARIABLES.DATA(2), width =>2);
   put(",");
   put(tuple.GLOBAL_VARIABLES.CONTROL(1), width =>2);
   put(",");
   put(tuple.GLOBAL_VARIABLES.CONTROL(2), width =>2);
   put(",");
   put(tuple.machine1_state.current,width=>1);
   put(" ]");
end output_Gtuple;

separate (main)
procedure output_Gtuple_to_file(tuple : in out Gstate_record_type;
                                counter : in out integer) is
begin
   put(reach,counter);
   put(reach,"   [" & integer'image(tuple.machine1_state.state_number));
   put(reach," ," & integer'image(tuple.machine2_state.state_number));
   put(reach,",");
   put(reach,tuple.machine1_state.out_buffer(1), width => 1);
   put(reach,",");
   put(reach,tuple.machine1_state.out_buffer(2), width => 1);
   put(reach,",");
   put(reach,tuple.machine1_state.ack_rec(1),width=>1);
   put(reach,",");
   put(reach,tuple.machine1_state.ack_rec(2),width=>1);
   put(reach,",");
   put(reach,tuple.machine2_state.in_buffer(1), width => 1);
   put(reach,",");
   put(reach,tuple.machine2_state.in_buffer(2), width => 1);
   put(reach,",");
   put(reach,tuple.machine2_state.pkt_rec(1),width=>1);
   put(reach,",");
   put(reach,tuple.machine2_state.pkt_rec(2),width=>1);
   put(reach,",");
   put(reach,tuple.GLOBAL_VARIABLES.DATA(1), width =>1);
   put(reach," ]");
   new_line(reach);
end output_Gtuple_to_file;

separate (main)
procedure output_Gstate_node(Gstate_pointer : in out Glink_type;
                             Error_flag     : in out boolean) is
begin
   output_line_count := output_line_count + 1;
   if ((output_line_count mod 10) = 0) then
      scroll_pause;
```

```
   end if;
   set_col(Gcolumn_set);
   put(Gstate_pointer.system_state_number, width => 3);
   output_Gtuple(Gstate_pointer.Gtuple);
   if ((Gstate_pointer.link1.Glink = null) and then (Gstate_pointer.link2.Glink = null)
and then
      (Gstate_pointer.link3.Glink = null) and then (Gstate_pointer.link4.Glink = null))
then
      Error_flag := true;
   else
      Error_flag := false;
   end if;
end output_Gstate_node;
```

OUTPUT

REACHABILITY ANALYSIS of : sel_rep_w2

Global State GRAPH

```
 0   [ 0 , 0,D1,D2,F,F,F,E ,E ,F,F,E  E ,E ,E ,1 ]      snd_data  1
 1   [ 1 , 0,D1,D2,F,F,F,E ,E ,F,F,D1 E ,E ,E ,2 ]      snd_data  2
                                                        rcv_data  3
 2   [ 3 , 0,D1,D2,F,F,T,E ,E ,F,F,D1 D2,E ,E ,2 ]      rcv_data  4
 4   [ 3 , 1,D1,D2,F,F,T,D1,E ,T,F,E  D2,E ,E ,2 ]      snd_ack   6
 6   [ 3 , 0,D1,D2,F,F,T,E ,E ,F,F,E  D2,A1,E ,2 ]      rcv_ack   8
                                                        rcv_data  9
 8   [ 4 , 0,D1,D2,T,F,T,E ,E ,F,F,E  D2,E ,E ,1 ]      adv_win1  1
                                                        rcv_data  10
10   [ 4 , 1,D1,D2,T,F,T,E ,D2,F,T,E  E ,E ,E ,1 ]      adv_win1  3
                                                        snd_ack   12
12   [ 4 , 0,D1,D2,T,F,T,E ,E ,F,F,E  E ,E ,A2,1 ]      adv_win1  5
 9   [ 3 , 1,D1,D2,F,F,T,E ,D2,F,T,E  E ,A1,E ,2 ]      rcv_ack   10
                                                        snd_ack   11
11   [ 3 , 0,D1,D2,F,F,T,E ,E ,F,F,E  E ,A1,A2,2 ]      rcv_ack   12
 3   [ 1 , 1,D1,D2,F,F,F,D1,E ,T,F,E  E ,E ,E ,2 ]      snd_data  4
                                                        snd_ack   5
 5   [ 1 , 0,D1,D2,F,F,F,E ,E ,F,F,E  E ,A1,E ,2 ]      rcv_ack   7
                                                        snd_data  6
 7   [ 2 , 0,D1,D2,T,F,F,E ,E ,F,F,E  E ,E ,E ,1 ]      adv_win1  0
```

System State GRAPH

```
 0   [ 0 , 0 ]   snd_data   [ 1 , 0 ]  1
 1   [ 1 , 0 ]   snd_data   [ 3 , 0 ]  2
                 rcv_data   [ 1 , 1 ]  3
 2   [ 3 , 0 ]   rcv_data   [ 3 , 1 ]  4
 4   [ 3 , 1 ]   snd_ack    [ 3 , 0 ]  6
 6   [ 3 , 0 ]   rcv_ack    [ 4 , 0 ]  8
                 rcv_data   [ 3 , 1 ]  9
 8   [ 4 , 0 ]   adv_win1   [ 1 , 0 ]  1
                 rcv_data   [ 4 , 1 ]  10
10   [ 4 , 1 ]   adv_win1   [ 1 , 1 ]  3
                 snd_ack    [ 4 , 0 ]  12
12   [ 4 , 0 ]   adv_win1   [ 1 , 0 ]  5
 9   [ 3 , 1 ]   rcv_ack    [ 4 , 1 ]  10
                 snd_ack    [ 3 , 0 ]  11
11   [ 3 , 0 ]   rcv_ack    [ 4 , 0 ]  12
 3   [ 1 , 1 ]   snd_data   [ 3 , 1 ]  4
                 snd_ack    [ 1 , 0 ]  5
 5   [ 1 , 0 ]   rcv_ack    [ 2 , 0 ]  7
                 snd_data   [ 3 , 0 ]  6
 7   [ 2 , 0 ]   adv_win1   [ 0 , 0 ]  0
```

Machine 1 Array Contents

From	To	Transition	Executed
0	1	snd_data	yes
1	2	rcv_ack	yes
1	3	snd_data	yes
2	0	adv_win1	yes
3	4	rcv_ack	yes
4	1	adv_win1	yes

Machine 2 Array Contents

From	To	Transition	Executed
0	1	rcv_data	yes
1	0	snd_ack	yes

REFERENCES

[BART 87] Barton, M., and others, "Architectural and Behavioral Modeling in Computer Communication," *Distributed Processing*, North-Holland, 1987.

[BENV 91] Benvenutti, Eugene S., "Selection and Specification of a Data Link Protocol for VSAT Based Inter-Lan Communications," *Master's Thesis*, U.S. Naval Postgraduate School, Monterey, Ca., 1991.

[BOOC 91] Booch, Grady, *Software Engineering with Ada*, Addison-Wesley Publishing Co., 1991.

[FELD 90] Feldman, M.B., *Data Structures with Ada*, Reston Publishing Co., 1990.

[GOUD 83] Gouda, M. G., "An Example for Constructing Communicating Machines by Stepwise Refinement," *Proc. 3rd IFIP WG 6.1 Int. Workshop on Protocol Specification, Testing, and Verification*, North-Holland Publ., 1983.

[GONZ 91] Gonzalez, Dean W., *Ada Programmer's Handbook and Language Reference Manual*, Benjamin/Cummings Publishing Co., 1991.

[HOLZ 91] Holzmann, Gerard J., *Design and Validation of Computer Protocols*, Prentice Hall Publishing Co., 1991.

[JENS 92] Jensen, Patricia B., "Historical Overview of Data Communication with Analysis of a Selective Repeat Protocol," *Master's Thesis*, U.S. Naval Postgraduate School, Monterey, Ca., 1992.

[KING 91] King, Paul W., "Formalization of Protocol Engineering Concepts," *IEEE Transactions on Computers*, April 1992.

[LUND 88] Lundy, G.M., and Miller, R.E., "A Variable Window Protocol Specification and Analysis," *Protocol Specification, Testing, and Verification*, VII, North-Holland, 1988.

[LUND 91a] Lundy, G.M., and Miller, R.E., "Specification and Analysis of a Data Transfer Protocol Using Systems of Communicating Machines," *Distributed Computing*, December 1991.

[LUND 91b] Lundy, G.M., "Using FDT's as a Teaching Tool in Computer Communications Networks,"submitted for publication.

[LUND 92a] Lundy, G. M. and Miller, R. E., "Analyzing a CSMA/CD Protocol Through a System of Communicating Machines Specification," submitted for publication.

[LUND 92b] Lundy, G. M., "Specification and Analysis of a Composition of Protocols," *Information Sciences*, to appear.

[MANB 89] Manber, Udi, *Introduction to Algorithms*, Addison-Wesley Publishing Co., 1989.

[PENG 91] Peng, Wuxu and Puroshothaman, S., "Data Flow Analysis of Communicating Finite State Machines," *ACM Transactions on Programming Languages and Systems*, Vol.13, No. 3, July 1991.

[RAND 92] Randall, M., "Communication Protocol Conformance Testing Using a System of Communicating Machines Specification," *Master's Thesis*, Naval Postgraduate School, to be published 1992.

[RUDI 83] Rudin, H. and West, C. H.,*Protocol Specification, Testing, and Verification, III*, Elsevier Science Publishers, B. V., 1983.

[RUMB 91] Rumbaugh, James and others, *Object-Oriented Modelling and Design*, Prentice Hall Publishing Co., 1991.

[SARI 91] Sarikaya, B. and others, "Estelle-based Test Generation Tool," *Computer Communications*, November 1991.

[SIST 91] Sisto, R. and others, "A Protocol for Multi-rendezvous of LOTOS Processes," *IEEE Transactions on Computers*, April 1991.

[SKAN 88] Skansholm, Jan, *Ada From the Beginning*, Addison-Wesley Publishing Co., 1988.

[STAL 91] Stallings, William, *Data and Computer Communications*, 3rd ed., Macmillan Publishing Co., 1991.

[TANE 81] Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall Publishing Co.,1981.

[VUON 83] Vuong, Son T. and Cowan, Donald D.," Reachability Analysis of Protocols with FIFO Channels," *Proceedings, ACM SIGCOM*, March 8-9, 1983.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. G. M. Lundy Code CS/Lu Assistant Professor, Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Dr. Amr Zaky Code CS/Za Assistant Professor, Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Defense Information Systems Agency Code: DO ATTN: COL Bowser 701 South Courthouse Road Arlington, VA 22204-2199	1
6.	Headquarters Total Army Personnel Command TAPC-OPF-SC ATTN:LTC Kehoe 200 Stovall Street Alexandria, VA 2332-0415	1
7.	CPT Matthew J. Rothlisberger 7200 Leawood Lane Charlotte, NC 28227	1

Thesis
R7818 Rothlisberger
c.1 A tool for automated validation of network protocols.

DUDLEY KNOX LIBRARY
3 2768 00036924 3